UNIVERSIDADE FEDERAL DE SÃO CARLOS

CENTRO DE CIÊNCIAS EXATAS E DE TECNOLOGIA

PROGRAMA DE PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

RESHAPE: MONTAGEM HÍBRIDA DE GENOMAS COM FOCO EM ORGANISMOS BACTERIANOS COMBINANDO FERRAMENTAS *DE NOVO*

HÉRIO ÊNIO DE SOUSA PAZ

ORIENTADOR: PROF. DR. HERMES SENGER

São Carlos – SP

Junho/2017

UNIVERSIDADE FEDERAL DE SÃO CARLOS

CENTRO DE CIÊNCIAS EXATAS E DE TECNOLOGIA

PROGRAMA DE PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

RESHAPE: MONTAGEM HÍBRIDA DE GENOMAS COM FOCO EM ORGANISMOS BACTERIANOS COMBINANDO FERRAMENTAS *DE NOVO*

HÉRIO ÊNIO DE SOUSA PAZ

Dissertação apresentada ao Programa de Pós-Graduação em Ciência da Computação da Universidade Federal de São Carlos, como parte dos requisitos para a obtenção do título de Mestre em Ciência da Computação, área de concentração: Sistemas de Computação
Orientador: Prof. Dr. Hermes Senger

São Carlos – SP

Junho/2017

UNIVERSIDADE FEDERAL DE SÃO CARLOS

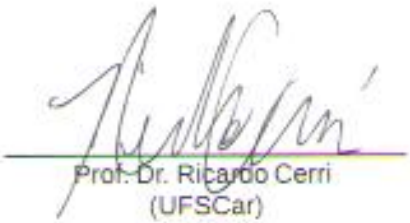
Centro de Ciências Exatas e de Tecnologia

Programa de Pós-Graduação em Ciência da Computação

Folha de Aprovação

Assinaturas dos membros da comissão examinadora que avaliou e aprovou a defesa de Dissertação de Mestrado do candidato Hério Ênio de Sousa Paz, realizada em 26/06/2017.

Prof. Dr. Hermes Senger
(UFSCar)

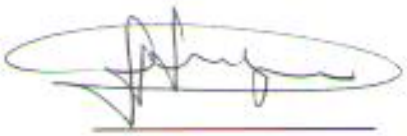


Prof. Dr. Ricardo Cerri
(UFSCar)

****

Prof. Dr. Fabricio Alves Barbosa da Silva
(UFU)

Certifico que a sessão de defesa foi realizada com a participação à distância do membro Fabricio Alves Barbosa da Silva, depois das arguições e deliberações realizadas, o participante à distância está de acordo com o conteúdo do parecer da comissão examinadora redigido no relatório de defesa do aluno Hério Ênio de Sousa Paz.



Prof. Dr. Hermes Senger
Presidente da Comissão Examinadora
(UFSCar)

AGRADECIMENTOS

Agradeço primeiramente a todas as pessoas que contribuiram de alguma forma para a realização deste trabalho, direta ou indiretamente.

Aos meus pais, Hélio e Erineide, que desde sempre me apoiaram e me incentivaram em todas as etapas dos meus estudos, meus avós, José Luís e Maria da Conceição, a quem mais deixei saudades, e à toda minha família: meu irmão Hélvis, minha tia Edna e meus tios Érico, Ernando e Ézio.

Aos mitos do 412: Elyson, Nihey e Eduardo, amigo de longa data do IRC e também discípulo do *Tio Grimço* (Prof. Dr. Eric Grimson), quem me recomendou a vir para São Carlos, me colocou no caminho do Jazz, algo que não consigo mais viver sem.

Aos meus amigos do mestrado Ana, Camila, Débora (minha *accountability buddy*), Fernanda e Flávio, que proporcionaram os momentos mais divertidos e engraçados durante o mestrado, dentro e fora do departamento, e aos demais colegas.

Ao Starkad Team: Jordão, Lincoln, Pablo, Rafael, Ruhan e Wermeson, que joga e programa unido (ou não), desde os tempos de graduação na UFPI.

E por fim, agradeço ao Prof. Dr. Hermes Senger e ao Prof. Dr. Fabrício A. B. da Silva, pela orientação e pelas contribuições no mestrado e na vida acadêmica, à CAPES e à Universidade Federal de São Carlos.

Bioinformatics is 90% converting format A to B, 5% reimplementing, and 5% actual science.

/u/xlrx02

RESUMO

O aumento da capacidade computacional dos computadores e a demanda por sequenciadores de baixo custo impulsionou o desenvolvimento de sequenciadores de nova geração com alta taxa de vazão de dados, fenômeno que expandiu exponencialmente o volume de dados de sequenciamento disponíveis para análise. Essas tecnologias fazem uso de diversos métodos que paralelizam o processo de sequenciamento, produzindo grandes quantidades de leituras simultaneamente. Algoritmos de montagem de genomas usualmente são desenvolvidos para trabalhar com tecnologias específicas de sequenciamento. No entanto, uma grande quantidade de genomas ainda não foram reunidas em um único fragmento. Com isso diversas abordagens híbridas, que utilizam mais de um tipo de leitura, vêm sendo desenvolvidas.

Um dos objetos de estudo deste trabalho é a bactéria *Pseudomonas aeruginosa* CCBH4851, um agente causador de infecções hospitalares, e ainda não tinha o seu genoma completamente montado. Motivado pelo problema de montagem desse genoma, este trabalho propõe o desenvolvimento do reSHAPE: um *pipeline* híbrido para montagem de genomas de bactérias, que possibilite tanto gerar novas montagens quanto melhorar montagens já existentes de diversos genomas, por meio da integração de diversos métodos e ferramentas *de novo*. A partir da montagem gerada pelo reSHAPE, foi possível finalizar o genoma da *P. aeruginosa* manualmente em apenas um fragmento. Além disso, o reSHAPE foi capaz de melhorar a atual montagem de outras bactérias, obtendo resultados superiores ao estado da arte dos montadores híbridos.

Palavras-chave: Sequenciamento, nova geração, genomas, montagem, bactéria, *pipeline*, *de novo*

ABSTRACT

Increased computing power of computers and the demand for low-cost sequencers boosted the development of Next-Generation Sequencers with high-rate data flow, phenomenon that exponentially expanded the volume of sequence data available for analysis. Those technologies use a variety of methods that parallelize the sequencing process, producing a high amount of reads concurrently. Genome assembly algorithms are usually developed in order to support specific sequencing technologies. However, a large number of genomes still were not combined in a single fragment. Thus, several hybrid approaches, that use more than one type of read, have been developed.

One object of study of this work was the bacterium *Pseudomonas aeruginosa* CCBH4851, an organism that causes hospital-acquired infections, which has not yet been fully assembled. Motivated by the problem of assembling its genome, this work proposes the development of reSHAPE: a hybrid genome assembly pipeline focused on bacterial organisms that allows to generate new assemblies as well as improve already existing assemblies of different genomes, by integrating several *de novo* methods and tools. From the reSHAPE assembly generated for *P. aeruginosa*, it was possible to manually complete its genome in a single fragment. Additionally, reSHAPE was able to improve the current assemblies of some bacteria, achieving superior results when compared to other state of the art hybrid assemblers.

Keywords: Sequencing, Next-Generation, genome, assembly, bacterium, pipeline, *de novo*

LISTA DE FIGURAS

LISTA DE TABELAS

.

GLOSSÁRIO

DBG – *De Bruijn Graph*

ENA – *European Nucleotide Archive*

MPI – *Message Passing Interface*

NCBI – *National Center for Biotechnology Information*

NGS – *Next-Generation Sequencing*

OLC – *Overlap-Layout-Consensus*

PacBio – *Pacific Biosciences*

ReSHAPE – *Sousa's Hybrid Assembly Pipeline*

SCSP – *Shortest Common Supersequence Problem*

SGE – *Sun Grid Engine*

SMP – *Symmetric Multi-Processing*

UML – *Unified Modeling Language*

UPC – *Berkeley Unified Parallel C*

SUMÁRIO

Capítulo 1

INTRODUÇÃO

1.1 Visão geral

Desde a década de 80, os dados provenientes de sequenciamento de genomas eram mantidos por grandes bases de dados como o *National Center for Biotechnology Information* (NCBI), o que permitia que os laboratórios de sequenciamento pudessem submeter os seus dados para armazenamento, e ao mesmo tempo, ter acesso a várias fontes com diversas outras sequências.

Isso tornava o trabalho dos profissionais de bioinformática mais simples, pois os mesmos eram capazes de baixar os dados que necessitavam, utilizá-los, e descartá-los quando deixassem de ser necessários (MEYER, 2006). Entretanto, em meados dos anos 2000, foram surgindo os sequenciadores de nova geração, que passaram a possibilitar a análise de genomas de maior comprimento, aumentando o número de genomas sequenciados.

Com as novas tecnologias de sequenciamento, a velocidade da geração de dados provenientes de sequenciamento já ultrapassou o poder de processamento dos computadores descrito pela Lei de Moore (JAGADISH et al., 2014). Em decorrência disso, novas tecnologias de sequenciamento e armazenamento de genomas vem sendo desenvolvidas, utilizando-se de técnicas de aprendizado de máquina, computação em nuvem, e principalmente, computação paralela (STEIN, 2010). Esses novos métodos tem caracterizado o que se chama de sequenciamento de nova geração (do inglês *Next-Generation Sequencing* (NGS)).

As mais recentes tecnologias de sequenciamento paralelo proporcionam altas taxas de *throughput* a um custo consideravelmente menor, proporcional aos dados. No entanto, os dados produzidos são sequências muito curtas provenientes de várias leituras feitas sobre o genoma original (METZKER, 2010), o que faz com que a montagem *de novo* seja extremamente desafiadora em relação a capacidade computacional.

O processo de sequenciamento inicia partindo-se fisicamente o DNA em fragmentos que variam entre 200 a 200.000 bases, de forma que não é mantida nenhuma informação de qual parte da sequência de DNA os fragmentos vieram. Um sequenciador lê de 1.000 a 10.000 bases no início e no fim dos fragmentos, produzindo pares de leituras, para então um *assembler* reconstruir a sequência original sobrepondo as leituras geradas pelo sequenciador. Essa técnica de sequenciamento é conhecida como *shotgun* (POP; SALZBERG; SHUMWAY, 2002).

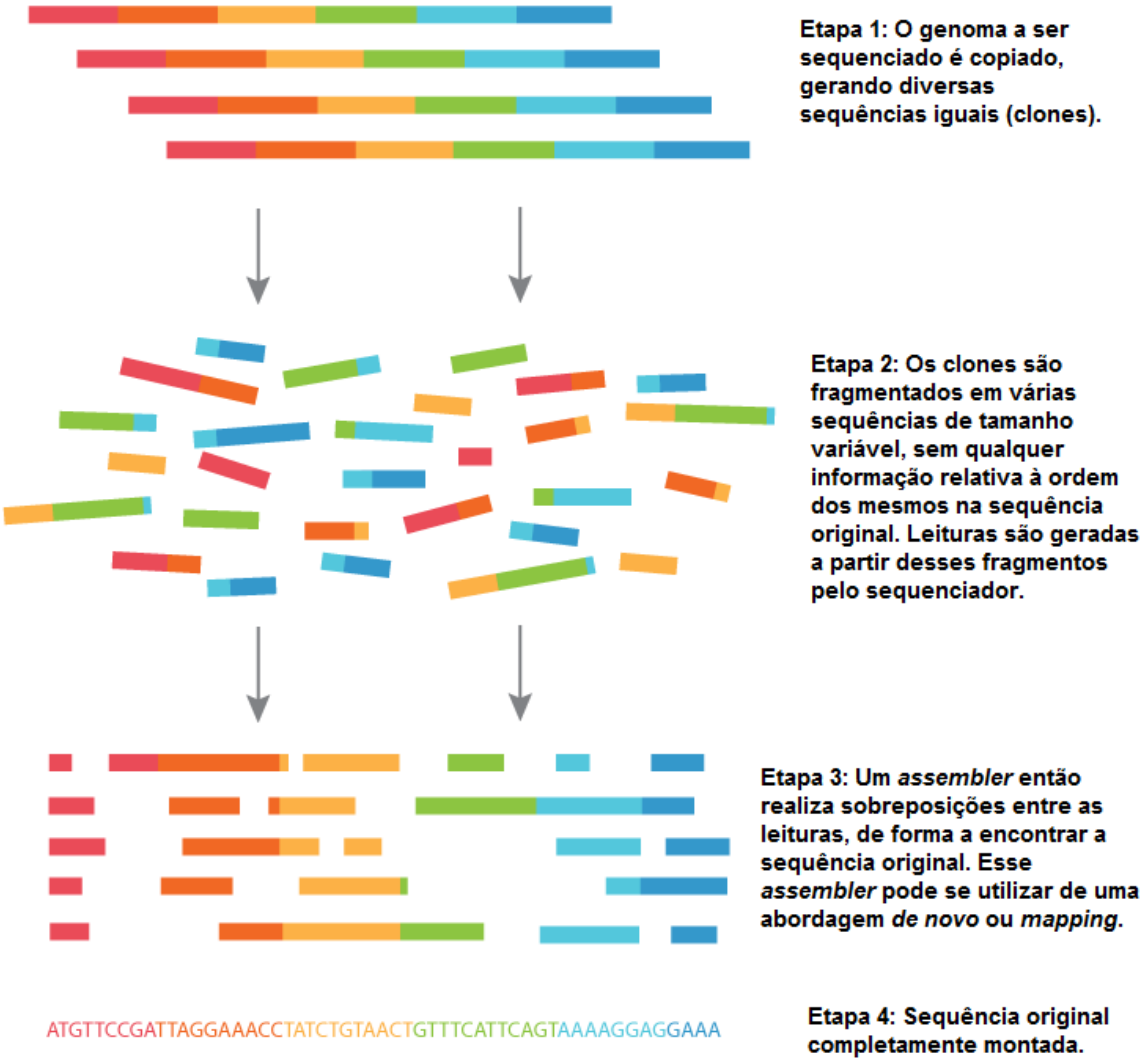


Figura 1.1: Técnica de *shotgun*
Adaptado de:
⟨https://www.abmgood.com/marketing/knowledge_base/next_generation_sequencing_data_analysis.php⟩

A tarefa realizada pelo *assembler* consiste em, após obter os fragmentos gerados pelo sequenciador, alinhar os mesmos de forma a encontrar sobreposições entre as leituras e assim reconstruir a sequência original. Para essa tarefa, existem dois tipos de abordagem: *de novo* e *mapping* (BAKER, 2012)(SCHBATH et al., 2012).

A abordagem *de novo* consiste em unir leituras de curto tamanho para criar sequências inteiras, muitas vezes completamente novas. Esse método faz com que não exista viés em relação a um genoma de referência, visto que não necessita se adaptar a um *template*. Além disso, o fato de utilizar leituras curtas faz com que a montagem seja mais fragmentada, funcionando melhor para diferenças de média ou larga escala entre os fragmentos.

A abordagem *mapping* utiliza uma sequência existente como base, gerando uma sequência que é semelhante, mas não necessariamente idêntica à sequencia base. Esse processo gera menos contíguos (do inglês *contigs*, são conjuntos de segmentos de DNA, que sobrepostos, representam uma região de consenso do DNA), entretanto, em vários métodos que realizam *mapping*, as leituras não mapeadas não são utilzadas na sequência final, da mesma forma que as novas sequências geradas que são completamente diferentes do genoma base também são perdidas. Devido as características de cada abordagem, metodos híbridos que combinam os dois métodos são utilizados (UTTURKAR et al., 2014).

Dentre as novas tecnologias de montagem *de novo*, destacam-se três tipos de algoritmos, todos baseados em grafos: métodos *overlap-layout-consensus* (OLC), que se baseiam em grafos de superposição, métodos de grafos De Bruijn (DBG), que utilizam uma representação *k*-mer, e algoritmos gulosos, que podem se utilizar das duas formas anteriores (MILLER; KOREN; SUTTON, 2010)(COMPEAU; PEVZNER; TESLER, 2011).

A representação de grafo De Bruijn é a mais adequada para montagem de genomas grandes (LI et al., 2012). Dado um alfabeto, o grafo de Bruijn é formado por todas as subsequências possíveis de tamanho *k* do mesmo. Diversas versões do mesmo foram propostas, de forma a diminuir a quantidade de memória necessária para armazená-lo (CHIKHI et al., 2015).

1.2 Motivação e objetivos

Em se tratando de NGS, as tecnologias de sequenciamento podem utilizar leituras longas ou curtas (QUAIL et al., 2012). Os sequenciadores de leituras curtas usualmente apresentam alto *throughput* a um baixo custo de sequenciamento por base, entretanto, dificultam a identificação do ponto de junção exato das sequências geradas pelas sucessivas leituras (HENSON; TISCHLER; NING, 2012), fazendo com que uma grande quantidade de informação seja descartada.

Quanto aos sequenciadores de leituras longas, tem como pontos positivos o tempo menor de execução, e a possibilidade de gerar múltiplas sequências por volta de 10.000 bp (pares de base). Entretanto, a taxa de erro de sequenciamento ainda é alta, o custo de base e do próprio equipamento são altos, e o *throughput* é moderado. Portanto, o uso de apenas uma tecnologia

de sequenciamento não é suficiente para realizar a completa montagem de diversos genomas, o que se extende aos algoritmos de montagem.

Dessa forma, novas estratégias híbridas vem sendo desenvolvidas para se aproveitar as vantagens de cada tipo de sequenciamento (DIMOND, 2012). Dentre elas podemos destacar técnicas que combinam leituras e montagens de provenientes de diferentes tecnologias de sequenciamento e algoritmos de montagem, sendo aplicadas em diversas tarefas, como montagens *de novo*, correção de erros de sequenciamento e melhoria da qualidade de sequências.

Esse trabalho tem como objetivo propor um *pipeline* que faça a integração de diversas ferramentas *de novo* para melhorar montagens já realizadas, diminuindo a quantidade de fragmentos de sequências presentes na montagem final.

1.3 Organização do trabalho

O Capítulo 2 se inicia tratando a respeito das principais tecnologias de sequenciamento, bem como as suas características, vantagens e desvantagens. Em seguida são apresentados os principais problemas de NGS e as estratégias comumente utilizadas para resolvê-los. As estruturas de dados que realizam a representação dos fragmentos de sequência em memória também são abordadas.

O Capítulo 3 apresenta uma visão sobre a montagem de genomas, falando sobre o processo de montagem e sobre como os montadores podem ser classificados, de acordo com diversos quesitos. Posteriormente, diversos montadores são discutidos, separados de acordo com o tipo de leituras que utilizam no processo de montagem, enquanto o Capítulo 4 percorre todo o processo de elaboração e desenvolvimento do *pipeline* proposto, desde o projeto inicial e os experimentos preliminares, até o formato atual da ferramenta.

Por sua vez, o Capítulo 5 apresenta os resultados obtidos pelo reSHAPE e os compara com outros montadores híbridos, utilizando tanto o estudo de caso, quanto outras bactérias. Logo após, esses mesmos resultados são postos diante das montagens atuais para cada uma das bactérias disponíveis no banco de nucleotídeos do NCBI. Por fim, o Capítulo 6 apresenta a conclusão deste trabalho.

Capítulo 2

NEXT-GENERATION SEQUENCING (NGS)

2.1 Tecnologias de sequenciamento

Como mencionado, as plataformas de sequenciamento de nova geração podem trabalhar com leituras curtas, de até 700 bp, ou com leituras longas, até 10.000 bp. Dentre elas, as principais são as tecnologias Illumina e Ion Torrent para leituras curtas, e Pacific Biosciences (PacBio) para leituras longas (QUAIL et al., 2012).

O principal sequenciador IonTorrent é o PGM, que realiza o sequenciamento utilizando semicondutores, fazendo com que seja o equipamento de menor custo e mais rápido entre as tecnologias citadas. Entretanto, sequenciadores IonTorrent apresentam problemas no sequenciamento de homopolímeros (sequências compostas por bases idênticas) pelo fato de não detectarem nucleotídeos diretamente.

Os sequenciadores Illumina utilizam uma abordagem de sequenciamento por síntese, e os principais são o GAIIx, o MiSeq, e o HiSeq. Em comparação com o IonTorrent PGM, os sequenciadores Illumina conseguem ter menor taxa de erro de sequenciamento (menos de 1%), suportando leituras de maior tamanho (700 contra 250 bp).

O custo dos equipamentos Illumina é relativamente alto, especialmente o HiSeq, que chega próximo ao valor dos sequenciadores de leituras longas. Entretanto, o MiSeq foi desenvolvido para ser mais barato, diminuindo-se a quantidade de bases sequenciadas por corrida.

A tecnologia PacBio desenvolveu uma estrategia de sequenciamento de molécula única em tempo real, que torna o processo de sequenciamento rápido (igualando o tempo de execução do PGM), ao passo que possibilita trabalhar com leituras da ordem 10.000 bp, o maior tamanho até o presente momento.

No entanto, o PacBio RS é o sequenciador mais caro entre as plataformas mencionadas. Além disso, possui um *throughput* moderado em relação aos sequenciadores Illumina/IonTorrent, e uma taxa de erro de sequenciamento considerada alta, de aproximadamente 13%.

2.2 Principais problemas de NGS

2.2.1 Alinhamento de sequências

O alinhamento de sequências consiste na identificação de regiões de similaridade que possam denotar relações evolutivas entre duas cadeias, comparando-se a amostra que se deseja sequenciar com a sua sequência de referência. Ao se sequenciar de forma completa um organismo, o seu genoma de referência permite verificar as diferenças genéticas entre outros organismos de mesma espécie (FLICEK; BIRNEY, 2009).

O problema do alinhamento de sequências pode ser descrito por, dadas duas sequências $Q = \{q_1, ..., q_m\}$ e $R = \{r_1, ..., r_m\}$, deve-se encontrar a distância (ou similaridade) entre as mesmas. Define-se a similaridade de duas sequências como o valor de seus alinhamentos de melhor *score*, onde o *score* de um alinhamento é calculado pela soma dos pares de letras alinhadas, e letras alinhadas com caracteres nulos.

Os alinhamentos entre sequências podem ser globais ou locais, onde são aplicados métodos computacionais baseados em programação dinâmica para resolvê-los. O método de alinhamento global consiste no algoritmo de Needleman-Wunsch (NEEDLEMAN; WUNSCH, 1970), onde tenta-se alinhar cada uma das bases de cada sequência, enquanto a técnica de alinhamento local utiliza-se do algoritmo de Smith-Waterman (SMITH; WATERMAN, 1981), mais adequado para localizar regiões isoladas de similaridade entre duas sequências.

Existem também métodos híbridos, que realizam um alinhamento chamado semi-global, destinados a encontrar determinados padrões dentro de uma sequência longa, usando o algoritmo de Needleman-Wunsch com modificações (ERICKSON; SELLERS, 1983). No contexto desse trabalho, o alinhamento de sequências é efetuado por ferramentas conhecidas como metamontadores, a serem exploradas na Seção 3.6.

2.2.2 Montagem de sequências

A montagem de sequências consiste basicamente em unir uma grande quantidade de curtas sequências de DNA de forma a gerar uma representação da sequência de DNA original com-

pleta da qual os fragmentos menores se originaram (BAKER, 2012). O principal desafio em resolver esse problema se encontra no fato de que muitos genomas contêm um grande número de sequências idênticas, conhecidas como *repeats*, que podem possuir milhares de nucleotídeos, e alguns deles ocorrem em diversos lugares da sequência, principalmente em genomas maiores como os de plantas e animais.

O problema de montagem de sequência é análogo ao problema da supercadeia mais curta em comum (em inglês, *Shortest Common Supersequence Problem* (SCSP)), onde o objetivo é, dado um conjunto de *strings* $S = \{s_1, ..., s_n\}$, encontrar a *string* de tamanho mínimo que seja uma *superstring* de todo $s_i \in S$. Em outras palavras, deve-se encontrar a menor *string* x tal que toda *string* s_i seja uma *substring* de x. Embora esse problema seja NP-difícil, existem algoritmos de aproximação eficientes (HALLETT, 1998).

Um desses algoritmos é a abordagem de algoritmos gulosos, que foi a primeira solução para o problema da montagem a ser implementada com sucesso pelos sequenciadores de nova geração. A estratégia utilizada pelo algoritmo guloso pode ser resumida em:

1. Calcular todos os possíveis alinhamentos par-a-par de todos os fragmentos e atribuir um *score* a cada sobreposição em potencial.
2. Escolher dois fragmentos com a maior sobreposição (maior *overlap score*).
3. Combinar (*merge*) os fragmentos escolhidos.
4. Repetir os passos 2 e 3 até que reste apenas um fragmento.

Algoritmos gulosos são simples e de fácil implementação, mas a sua natureza local faz com que possam efetuar cálculos repetitivos, resultando em um aumento da memória RAM necessária para sua excecução, limitando o seu uso ao sequenciamento de genomas de organismos como bactérias e seres eucariontes unicelulares. Com isso, foram desenvolvidos os métodos de *Overlap-Layout-Consensus* e de grafo De Bruijn, que passam a enxergar o genoma como um grafo (POP; SALZBERG; SHUMWAY, 2002) (MILLER; KOREN; SUTTON, 2010).

Para entender como essas abordagens representam o genoma em forma de grafo, deve-se primeiro falar sobre k-mers. O termo k-mer se refere a todas subcadeias possíveis de tamanho k contidas em uma string. Considerando que as leituras realizadas por um sequenciador possuem o mesmo tamanho, pode-se assumir que as leituras são k-mers para algum valor k, e cada k-mer é gerado a partir de uma única leitura. Dada uma string s, a sua composição k-mer é o conjunto de todas as subcadeias k-mer de s em ordem lexicográfica, por exemplo, $Composition_3(ATACCCCACG) = \{ACC, ACG, ATA, CAC, CCA, CCC, CCC, TAC\}$.

O problema de composição de strings pode ser enunciado como: a partir de um inteiro k e uma string s, deve-se gerar $Composition_k(s)$. Dessa forma, para montar um genoma, o problema inverso (problema de reconstrução de strings) precisa ser resolvido: a partir de um inteiro k e um conjunto P de k-mers, deve-se gerar uma string que possua composição k-mer igual ao conjunto P.

Uma forma simples de resolver esse problema seria conectar um par de k-mers se eles se sobrepõem em $k-1$ letras. Entretanto, k-mers repetidos fazem com que exista mais de um caminho a se seguir para montar uma sequência.

2.3 Representação de sequências em memória

2.3.1 Grafo de sobreposição

O método de *Overlap-Layout-Consensus* se utiliza de um grafo de sobreposição (*overlap graph*), que representa as leituras (k-mers) como vértices. A composição k-mer de uma sequência consiste dos k-mers da mesma na ordem do genoma original (Figura 2.1), sendo que o sufixo do vértice anterior deve ser igual ao prefixo do próximo.

Portanto, para se efetuar a reconstrução do genoma, deve-se encontrar um caminho no grafo de sobreposição: uma aresta é direcionada de um vértice A a um vértice B quando o $k-1$ sufixo do primeiro se sobrepõe ao $k-1$ prefixo do segundo. A partir de um conjunto de k-mers, pode-se gerar um grafo de sobreposição (Figura 2.2), representado por uma lista de adjacência.

Para encontrar o genoma original, deve-se encontrar um caminho nesse grafo que passe por cada vértice apenas uma vez, em outras palavras, um caminho hamiltoniano. Entretanto, o problema de se encontrar um caminho hamiltoniano em um grafo é NP-completo, ou seja, não se conhece um algoritmo eficiente para resolver esse problema.

Por isso, um novo método foi desenvolvido para melhor representar uma composição k-mer, chamada de grafo De Bruijn. A próxima subseção se propõe a apresentar o grafo De Bruijn e as suas variações (PEVZNER; TANG; WATERMAN, 2001).

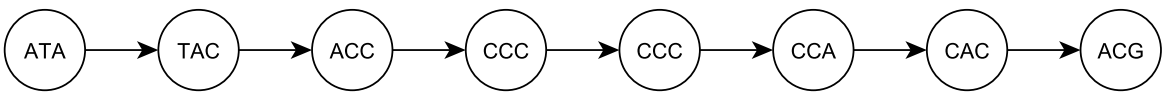


Figura 2.1: Composição k-mer da sequência ATACCCCACG em um grafo de sobreposição com $k = 3$

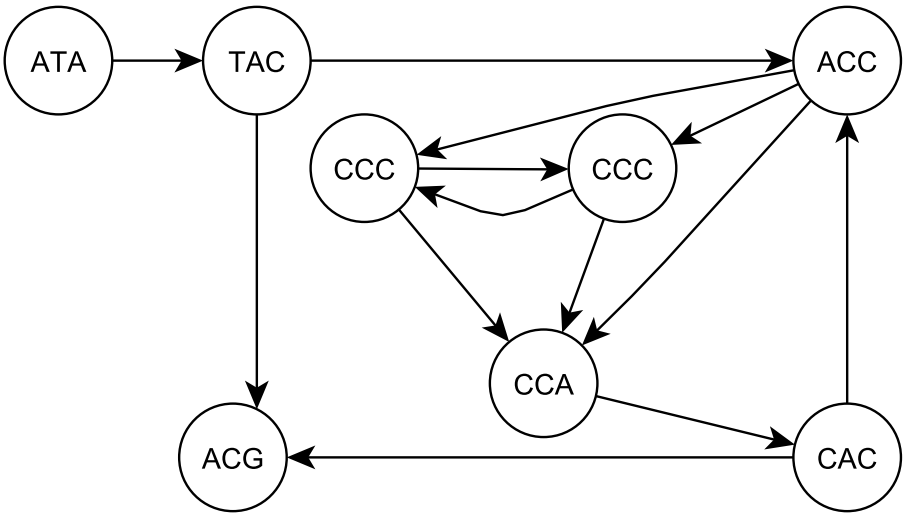


Figura 2.2: Grafo de sobreposição da sequência ATACCCCACG com $k = 3$

2.3.2 Grafo De Brujin

O grafo De Bruijn também é utilizado para se representar uma composição *k*-mer. Diferentemente do grafo de sobreposição, que representa os *k*-mers como vértices, o grafo De Bruijn representa os mesmos como arestas. Dessa forma, podemos obter os rótulos dos vértices acrescentando os prefixos e sufixos de cada *k*-mer (aresta) nos vértices origem e destino, respectivamente. Além disso, vértices idênticos (substrings *k*-mer que aparecem mais de uma vez) de uma sequência são unidos em um único vértice, e as arestas de cada um dos vértices unidos são adicionados ao novo vértice.

Na abordagem De Bruijn, a composição *k*-mer de uma string *s* é o grafo que possui $|s| - k +$ 1 arcos (arestas direcionadas) isolados, sendo cada um dos arcos rotulados por um *k*-mer. Como dito anteriormente, cada arco conecta dois vértices, sendo o primeiro rotulado pelo prefixo do *k*-mer, e o segundo pelo sufixo.

Gerar o grafo De Bruijn a partir da composição *k*-mer (Figura 2.3) consiste em unir os seus vértices idênticos (Figura 2.4). Para se reconstruir a sequência, deve-se caminhar o grafo de forma a visitar cada aresta/arco apenas uma vez, ou seja, encontrar um caminho euleriano (MEDVEDEV et al., 2007). Ao contrário do problema de encontrar um caminho hamiltoniano em um grafo, existem algoritmos eficientes para encontrar um caminho euleriano, como o algoritmo de Hierholzer.

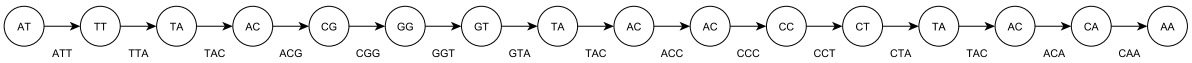


Figura 2.3: Composição k-mer da sequência ATTACGGTACCCTACAA em um grafo De Bruijn com $k = 3$

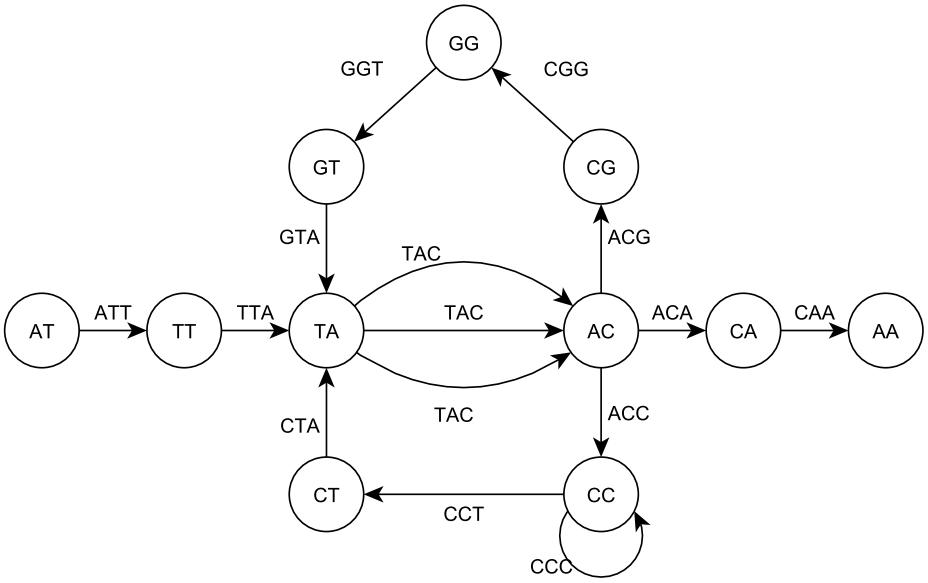


Figura 2.4: Grafo De Bruijn da sequência ATTACGGTACCCTACAA com $k = 3$

2.3.3 Grafo De Brujin pareado

À medida que se aumenta o tamanho da leitura (valor de k), menos vértices o grafo De Bruijn irá possuir, e consequentemente, se reduz o número de *repeats* (k-mers iguais). No entanto, também aumenta a dificuldade em achar *matches* para leituras com valor de k muito grande (SAMANTA, 2012).

A solução foi criar uma leitura (k-mer + d + k-mer), formada por um k-mer conhecido, uma região desconhecida de tamanho d e outro k-mer também conhecido. Isso permite o aumento do tamanho da leitura, diminuindo a probablidade de *repeats* (emaranhados no grafo De Bruijn). Essa abordagem é conhcida como (k,d)-mer.

Dada uma string s, um (k,d)-mer é um par de k-mers em s separados pela distância d. Utiliza-se a notação $(kmer1|kmer2)$ para representar um (k,d)-mer. Por exemplo, $(TAG|TGT)$ é um $(3,3)$-mer da sequência $GTAGAGCTGT$. A composição (k,d)-mer de uma string s é dada por $Composition_{k,d}(s)$, formada pelo conjunto de todos os (k,d)-mers de s, incluindo-se os repetidos, em ordem lexicográfica. Exemplo: $Composition_{3,1}(GTAGAGCTGT) = \{(AGA|CTG), (GAG|TGT), (GTA|AGC), (TAG|GCT)\}$.

A abordagem de (k,d)-mer foi incorporada ao grafo De Bruijn, criando-se o grafo De Bruijn pareado (MEDVEDEV et al., 2011). Dado um (k,d)-mer $(a_1,...,a_k|b_1,...,b_k)$, pode-se definir seu prefixo como o $(k-1,d)$-mer $(a_1,...,a_{k-1}|b_1,...,b_{k-1})$, e o seu sufixo como o $(k-1,d)$-mer $(a_2,...,a_k|b_2,...,b_k)$. Por exemplo, a partir do (k,d)-mer $(GTA|AGC)$ obtém-se o seu prefixo $(GT|AG)$ e o seu sufixo $(TA|GC)$.

No grafo De Bruijn pareado, dois (k,d)-mers são consecutivos se o prefixo do primeiro é igual ao sufixo do segundo. A composição k-mer de uma sequência s no grafo é o caminho formado por $|s|-(k+d+k+1)$ arcos correspondentes a todos os (k,d)-mers de s. Cada arco é rotulado por um (k,d)-mer, e os vértices iniciais e finais da arco são rotulados pelo prefixo e sufixo do mesmo, respectivamente, como mostrado pela Figura 2.5.

Análogo ao grafo De Bruijn tradicional, os vértices idênticos da composição (k,d)-mer também são unidos ao se gerar o grafo (Figura 2.6), e a sequência original é encontrada resolvendo o problema de encontrar um caminho euleriano em um grafo.

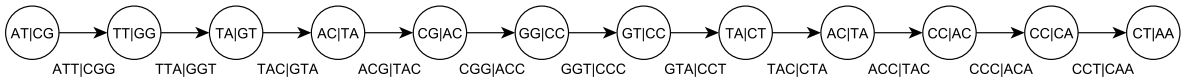


Figura 2.5: Composição k-mer da sequência ATTACGGTACCCTACAA em um grafo De Bruijn pareado com $k=3$ e $d=1$

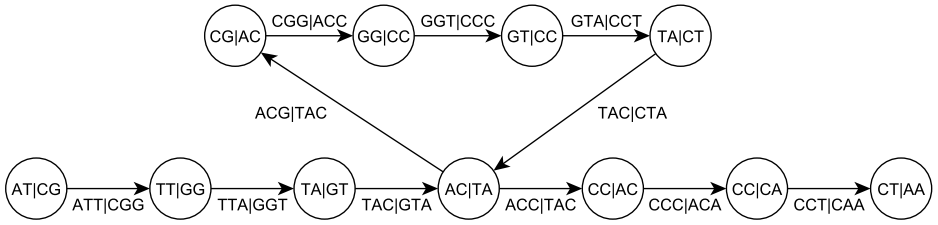


Figura 2.6: Grafo De Bruijn pareado da sequência ATTACGGTACCCTACAA com $k=3$ e $d=1$

2.3.4 Filtro de Bloom

Filtro de Bloom é uma esturura de dados probabilística, usada para testar se um elemento está em um conjunto (BLOOM, 1970). A estrutura pode permitir falsos positivos, mas não falsos negativos. Um filtro de Bloom consiste de um *array* de n bits, inicialmente com 0 em todas as posições, e h funções *hash*.

A adição de um elemento é feita computando cada um dos h valores *hash* do mesmo, ocupando h posições do array, e alterando os valores das mesmas para 1. Um elemento definitivamente não está no conjunto se algum bit dessas h posições é 0. Caso contrário (todos os bits

são 1), o elemento pode ou não estar no conjunto, pois os bits podem possuir valor 1 devido à inserção de outros elementos, o que resultaria em um falso positivo.

O filtro de Bloom pode ser utilizado para armazenar um grafo De Bruijn, com o objetivo de reduzir o espaço ocupado pelo grafo na memória. A inserção de todos os nós do grafo em um filtro de Bloom resulta em uma estrutura chamada grafo De Bruijn probabilistico (PELL et al., 2012).

Nesse grafo, as arestas são obtidas verificando se todas as possíveis extensões de um k-mer estão presentes no filtro. Uma extensão de um k-mer é o seu $k-1$ sufixo concatenado com uma das bases A, C, T, G ou uma delas concatenada com o seu $k-1$ prefixo. Devido ao filtro de Bloom, o grafo De Bruijn probabilistico pode permitir a existência de falsos nós, o que resultaria em falsas ramificações no grafo, e portanto, devem ser removidas após a construção do mesmo.

2.4 Formatos de arquivos em NGS

2.4.1 FASTA

O formato FASTA (LIPMAN; PEARSON, 1985), indicado pelas extensões .fa ou .fasta, é um arquivo de texto simples que contem um conjunto de sequências, representadas por um cabeçalho de uma linha, iniciado pelo caractere '>', contendo o nome e alternativamente uma descrição da sequência, seguido das bases que formam a sequência, como mostrado na Figura 2.7.

```
>seq1 Turkey
AAGCTNGGGCATTTCAGGGTGAGCCCGGGCAATACAGGGTAT
>seq2 Salmo gair
AAGCCTTGGCAGTGCAGGGTGAGCCGTGGCCGGGCACGGTAT
>seq3 H. Sapiens
ACCGGTTGGCCGTTCAGGGTACAGGTTGGCCGTTCAGGGTAA
>seq4 Chimp
AAACCCTTGCCGTTACGCTTAAACCGAGGCCGGGACACTCAT
>seq5 Gorilla
AAACCCTTGCCGGTACGCTTAAACCATTGCCGGTACGCTTAA
```

Figura 2.7: Sequências em um arquivo FASTA

Arquivos FASTA são usados para representar as montagens produzidas pelos *assemblers*, que podem ser compostas de contíguos, *scaffolds* ou ambos. Esses tipos de fragmentos serão melhor abordados na Seção 3.1. Além de sequências de DNA ou RNA, o formato FASTA também pode ser usado para representar sequências de aminoácidos em proteínas.

Os arquivos FASTA também podiam representar a qualidade das bases que compunham as sequências. Para isso, um arquivo FASTA adicional (também chamado de arquivo QUAL) ao arquivo de sequências deve conter os cabeçalhos na mesma ordem, com a informação de qualidade das bases representados por valores inteiros ao invés das sequências.

2.4.2 FASTQ

O formato FASTQ (COCK et al., 2009), indicado pelas extensões .fq ou .fastq é um formato geralmente usado para representar leituras geradas por sequenciadores, desenvolvido para armazenar as sequências e a qualidade das bases que a compõem no mesmo arquivo. Após o processo de sequenciamento, podem ser gerados um ou dois arquivos FASTQ, se o sequenciamento foi feito em uma ou nas duas direções do genoma.

Em um arquivo FASTQ, uma leitura é representada por quatro linhas, como na Figura 2.8:

```
@seq1 description 1
AACACCAAACTTCTCCACCACGTGAGCTACAAAAG
+
````Y^T]`]c^cabcacc`^Lb^ccYT\T\Y\WF
@seq2 description 2
TATGTATATATAACATATACATATATACATACATA
+
]KZ[PY]_[YY^```ac^\\`bT``c`\aT``bbb
@seq3 description 3
AACACCAAACTTCTCCACCACGTGAGCTACAAAAGGGT
+seq3 description 3
""Y^T]']C^CABCACC`^LB^CCYT\T\Y\WF^^^
@seq4 description 4
TATGTATATATAACATATACATATATACATACATA
+
]KZ[PY]_[YY^'"AC^\\'BT"C'\AT"BBB
```

Figura 2.8: Leituras em um arquivo FASTQ
```

1. Cabeçalho da sequência, iniciado pelo caractere '@', com identificador e descrição da sequência, sendo os dois últimos opcionais.

2. Bases da sequência pertencente à leitura.

3. Cabeçalho dos valores de qualidade, iniciado pelo caractere '+', opcionalmente seguido de identificador e descrição da sequência ou outros comentários.

4. Os valores de qualidade de cada uma das bases, representados por caracteres ASCII.

2.4.3 Outros formatos

Além dos formatos citados, há outros formatos usados para finalidades distintas, como por exemplo os arquivos SAM/BAM para representar alinhamentos. Ambos são bastante usados para armazenar mapeamentos de sequências a uma determinada sequência de referência. Enquanto o SAM armazena esses alinhamentos em texto simples, o BAM é a representação comprimida de forma binária do SAM.

Também podemos destacar o formato GFF e suas diversas versões, usado em anotação genômica. Esse processo consiste em identificar genes, regiões gênicas, e as suas funções em determinado genoma. Para tal, arquivos GFF armazenam descrições a respeito de genes e outras características do DNA, RNA ou de proteínas. No desenvolvimento deste trabalho, apenas dados proveniente de arquivos FASTA e FASTQ foram utilizados.

2.5 Considerações finais

Neste capítulo foram apresentados tópicos relativos ao sequenciamento de nova geração: suas principais tecnologias, problemas computacionais, e estruturas de dados comumente utilizadas em possíveis soluções para esses problemas.

Com relação as estruturas de dados, abordagens de grafo de sobreposição (OLC) e grafo De Bruijn (DBG) são usadas por diversas ferramentas para representar sequências em diversas técnicas de montagem. Por exemplo, OLC é frequentemente aplicado em mapeamento e combinação de sequências e até mesmo em alguns métodos de alinhamento, ao passo que DBG é mais usado para se produzir novas montagens a partir de leituras.

Uma das formas de se modificar o grafo de Bruijn também foi apresentada. Várias ferramentas que fazem uso de DBG costumam efetuar as suas próprias modificações, seja para

desenvolver novos algoritmos, ou para melhor representá-los em memória. Em termos de armazenamento, o filtro de Bloom é uma alternativa às tabelas *hash*, bastante usadas para armazenar sequências.

Ao final do capítulo, foi discorrido a respeito dos principais formatos de arquivos em NGS, suas características e especificações. O capítulo a seguir discorre sobre montagem de genomas, ao passo que fornece uma visão mais aprofundada acerca de diversas ferramentas de montagem que fazem uso das técnicas e estruturas apresentadas, além das suas próprias, e analisa como as mesmas funcionam na prática.

Capítulo 3

MONTAGEM DE GENOMAS

3.1 Processo de montagem

As técnicas de montagem mais comuns utilizam grafos OLC, DBG ou ambos. Usualmente, o processo de montagem de genomas baseado em grafos pode ser generalizado em quatro etapas (MILLER; KOREN; SUTTON, 2010), brevemente descritas na Figura 3.1. A primeira etapa consiste em gerar *k*-mers a partir de leituras, além de obter dados relativos à frequência dos *k*-mers em cada sequência e informação de *linkage* (adjacência de *k*-mers) para cálculo de heurísticas, dependendo da abordagem.

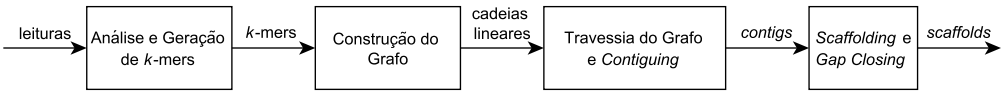


Figura 3.1: Etapas da montagem de genomas baseada em grafo

Na etapa seguinte, o grafo é gerado a partir das informações anteriormente obtidas. Cada *assembler*, seja sequencial ou em paralelo, possui diferentes abordagens e estruturas para representar os vértices e as arestas, armazená-los e percorrer o grafo para geração de contíiguos. Ainda nessa etapa, é realizada a simplificação do grafo e remoção de erros, como por exemplo, bolhas no grafo (do inglês *bubbles*, são caminhos distintos no grafo que iniciam e terminam nos mesmos nós) e caminhos de baixa cobertura.

O grafo gerado é então percorrido na etapa de travessia, onde os *k*-mers presentes no caminho percorrido são alinhados, de forma a se gerar contíguos, que são sequências que representam regiões do genoma que não se sobrepõem. Na etapa final, sequências maiores (*scaffolds*, figura 3.2) são reconstruídas, compostas de contíguos e lacunas (*gaps*), representadas pela letra N nos arquivos de saída.

Gaps ocorrem no genoma quando leituras presentes nas duas extremidades de um fragmento se sobrepõem com outras leituras em dois contíguos distintos. Pelo fato dos fragmentos serem de tamanho aproximadamente conhecido, o tamanho dos gaps entre os contíguos podem ser estimados.

Técnicas de *gap closing* são então utilizadas para remover as lacunas e gerar o genoma. No melhor caso, o genoma será representado por apenas um *scaffold*, entretanto, se não for possível reconstruí-lo inteiramente a partir das leituras iniciais, ele pode ser representado por vários *scaffolds*.

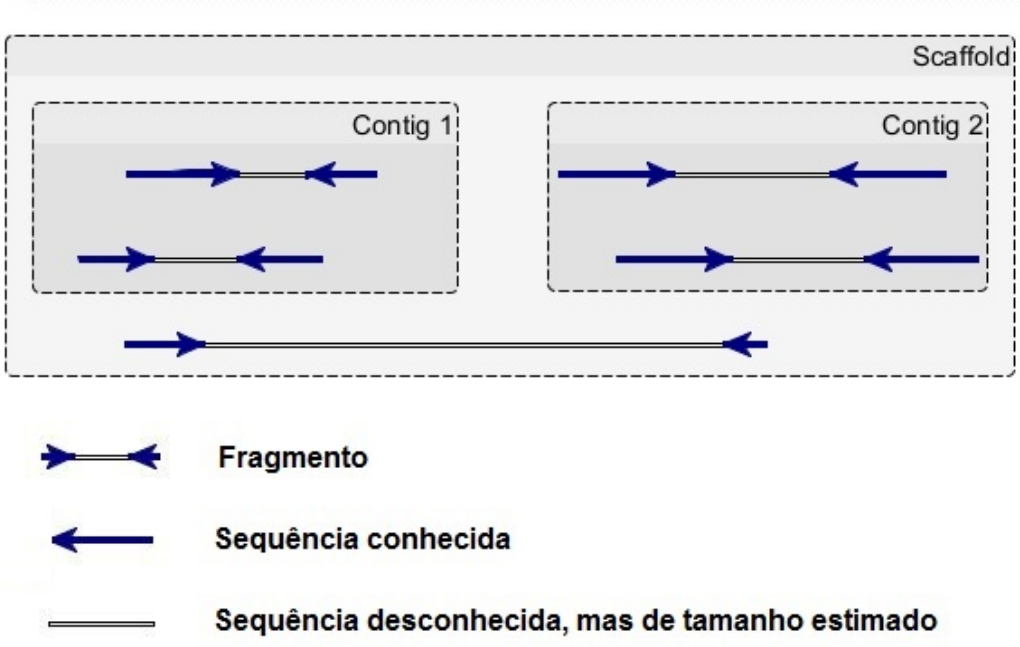


Figura 3.2: Representação de *scaffold*
Adaptado de: ⟨http://genome.jgi.doe.gov/help/scaffolds.jsf⟩

A Figura 3.3 mostra o resultado do processo de *scaffolding* em um conjunto de contíguos, bem como contíguos e *scaffolds* podem ser representados em arquivo. Nesse conjunto, após a fase de *scaffolding*, os contíguos 1 e 3 (destacados em azul) foram reunidos no mesmo fragmento (scaffold_1), com um gap de tamanho 8 bp (representado em vermelho), similar ao que aconteceu com os contíguos 2 e 4, reunidos no scaffold_2.

3.2 Classificação dos montadores

Usualmente os montadores são classificados de acordo com a abordagem que seus algoritmos utilizam para a montagem (MILLER; KOREN; SUTTON, 2010). Essas abordagens podem ser *Overlap-Layout-Consensus*, grafo De Bruijn ou algoritmos gulosos, que podem fa-

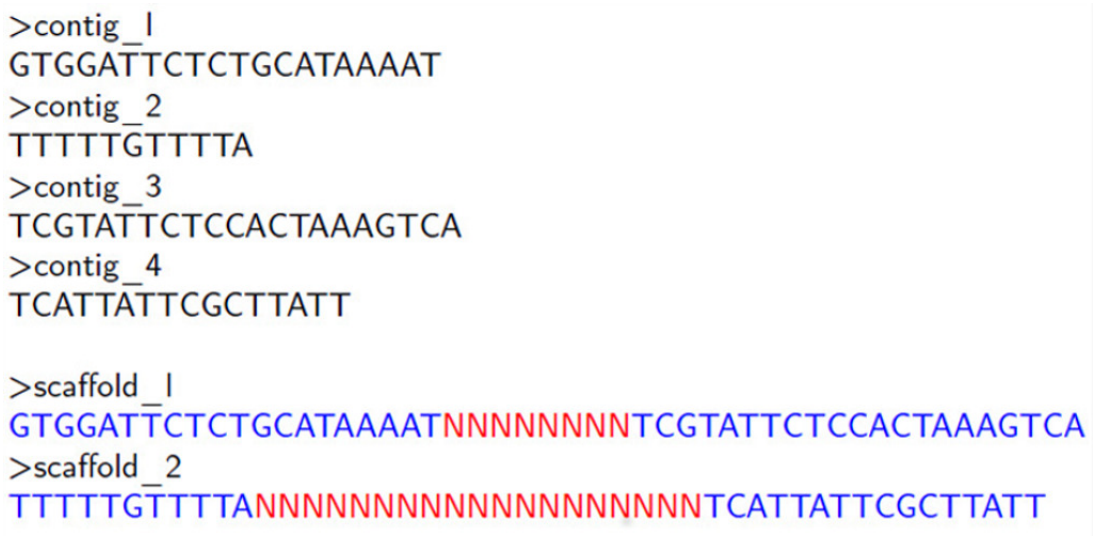


Figura 3.3: Representação de contíguos e *scaffolds* em arquivo

zer uso das anteriores. Um exemplo é o Ray (BOISVERT; LAVIOLETTE; CORBEIL, 2010), um algoritmo guloso executado sobre um grafo De Bruijn.

Ao passo que as novas tecnologias de sequenciamento possibilitaram o uso de leituras cada vez maiores, montadores foram desenvolvidos para trabalhar com os dados provenientes desses sequenciadores. Com isso, podemos apontar uma nova classificação, de acordo com o tipo de leitura que um montador pode utilizar.

Seguindo esse critério, um montador pode ser de leituras curtas, ou seja, com leituras provenientes de sequenciadores Illumina/IonTorrent e/ou anteriores (454, SOLiD, Sanger); leituras longas, produzidas por sequenciadores PacBio (e/ou Nanopore); ou híbridos, combinando os dois tipos de leituras para gerar grafos de consenso ou utilizando leituras longas para realizar *scaffolding* sobre contíguos gerados a partir de leituras curtas.

Vale ressaltar que não há um número exato que determine se uma leitura pode ser considerada longa ou curta. Geralmente sequenciadores produzem leituras possuem menos de 1.000 bp, enquanto leituras longas são usualmente maiores que 5.000 bp.

Existem também ferramentas que realizam montagens sem necessitar de qualquer tipo de leitura, baseando-se somente em montagens e outras informações acerca do próprio genoma a ser remontado, em uma abordagem chamada de meta-montagem. As próximas seções são dedicadas a apresentar os principais montadores em relação às leituras que utilizam.

3.3 Montadores de leituras curtas

3.3.1 ABySS

(SIMPSON et al., 2009) apresenta o ABySS, um *assembler* que realiza montagem de genomas em paralelo. O algoritmo do ABySS possui duas fases: a primeira consiste em gerar os *k*-mers a partir das leituras, onde são posteriormente processados de forma a remover erros provenientes das leituras, e contíguos iniciais são construídos. O ABySS utiliza uma representação distribuída do grafo De Bruijn. Nela, a localização de um *k*-mer pode ser computada a partir de sua sequência, e sua informação de adjacência é armazenada de forma independente da localização do *k*-mer.

Nessa representação as sequências estão distribuídas por vários nós. A localização de um dado *k*-mer é computável deterministicamente a partir de sua sequência. Uma função *hash* computa a localização de cada *k*-mer. Valores $\{0, 1, 2, 3\}$ são designados às bases $\{$A, C, G, T$\}$, e o valor *hash* é computado a partir desse valor numérico. O mesmo procedimento é realizado com o complemento reverso da sequência, e os dois valores são combinados por uma operação XOR nas suas representações em bit. O índice do nó que receberá o *k*-mer é calculado pelo resto da divisão do valor resultante da operação XOR pelo número de nós utlizados.

O ABySS utiliza uma representação compacta de arestas para armazenar a informação de adjacência entre *k*-mers. Cada vértice do grafo pode possuir até 8 arestas, correspondentes a cada extensão possível (A, C, G, T) nas duas extremidades da sequência. Utiliza-se 8 bits por *k*-mer para armazenar essa informação, onde um bit representa presença ou ausência de cada aresta. Os *k*-mers adjacentes podem ser facilmente gerados, e a sua localização pode ser computada pelo método descrito anteriormente.

Os dados são inicialmente carrgados no grafo De Bruijn distribuído (sequências de bases desconhecidas são descartadas). Cada sequência de entrada com tamanho l é quebrada em $l-k+1$ *k*-mers. O índice do nó que receberá o *k*-mer é computado e o mesmo é armazenado localmente nesse nó em uma tabela *hash*. Se o complemento da sequência de um *k*-mer já estiver presente na tabela, o *k*-mer não é armazenado.

Após todos os *k*-mers serem gerados e armazenados no grafo, as suas adjacências são computadas. Para cada *k*-mer presente nas sequências de entrada uma mensagem é enviada para cada um de seus oito possíveis vizinhos. Se o vizinho existir, a informação de vizinhança é atualizada de acordo. Bolhas e ramos "sem saída" do grafo são removidos, e o grafo é simplificado.

Na segunda fase, contíguos são extendidos até que se encontrem ambiguidades (pode-se estender o contíguo de mais de uma forma) ou uma extremidade. Após a etapa de extensão, A informação de adjacência é então utilizada para identificar contíguos que podem ser combinados. Essa informação é obtida na etapa anterior, onde para cada k-mer presente em um nó, uma mensagem é enviada para os seus oito possíveis vizinhos (quatro bases à esquerda e quatro bases à direita), onde um vizinho de um k-mer possui uma sobreposição de tamanho $k-1$ com o mesmo.

Dois contíguos C e P podem ser combinados se há um caminho único (uma sequência de contíguos) no grafo De Bruijn partindo-se de C e que visite cada contíguo presente em P (ou o oposto). O processo é repetido para cada contíguo, e o passo final une os caminhos para gerar os contíguos finais.

O ambiente de paralelismo no ABySS consiste de 21 nós com 16 GB de RAM e dois processadores quad-core Intel Xeon 2.66 GHz, totalizando 168 cores. Esses nós são conectados a um *switch* de 1 gigabit. O *software* é implementado em C++, e foi testado com ambos LAM/MPI 7.1.4 e Open MPI 1.2.6. O *cluster* utiliza a Sun Grid Engine (SGE) para escalonamento de *jobs*.

Os autores realizaram dois experimentos: o primeiro com tamanho de leitura 200 bp e $k = 36$, e o segundo com leituras de tamanho entre 36 e 42 bp e $k = 27$, utilizando genoma humano (NA18507 Yoruba individual). Também foi feito montagem do genoma da bactéria *Escherichia coli* (*E. coli* K12 MG1655), com $k = 31$ e tamanho de leitura 36 bp.

3.3.2 Ray

(BOISVERT; LAVIOLETTE; CORBEIL, 2010) apresenta o Ray, um algoritmo que combina as abordagens de grafo De Bruijn e de algoritmos gulosos para gerar sequências por meio de heurísticas, ao invés de percorrer o grafo De Bruijn em busca de caminhos eulerianos. A representação de grafo De Bruijn do Ray leva em consideração o conjunto de leituras e seus respectivos complementos reversos (D_+), o valor de k, e um valor de corte c.

Na etapa de obtenção de k-mers, uma função f_{D+} associa a cada valor c o número de k-mers que aparecem c vezes em D_+. Essa função possui um mínimo local, c_{min}, que estima o valor no qual há alta probabilidade de que a quantidade de dados incorretos é menor do que a de corretos. Também possui um máximo local, c_{peak}, que estima a cobertura média do genoma.

O conjunto de vértices do grafo é composto por todos os k-mers que aparecem no mínimo c vezes no conjunto D_+, enquanto o conjunto de arcos é formado por todos os $(k+1)$-mers que aparecem pelo menos uma vez em D_+. Leituras onde o primeiro k-mer com valor que

ocorrem pelo menos c vezes nas mesmas (x_r k-mers) possui um valor de confiança maior que 255 são removidas de D_+, pois o algoritmo as considera como *repeats*. Para os x_r restantes, são armazenadas anotações com a posição em que iniciam em cada uma das leituras.

Após a construção do grafo, o algoritmo determina algumas *seeds*, que são os caminhos máximos no grafo com valor de corte $\frac{c_{min}+c_{peak}}{2}$, que são compostos por vértices com ambos grau de saída e entrada de até um. O algortimo tenta estender cada uma das *seeds* aplicando duas heurísticas que decidem qual arco inserir no caminho, levando em consideração a distribuição das leituras na *seed* em questão, e funções de distância. Mais detalhes sobre as regras que compõem as heurísticas em (BOISVERT; LAVIOLETTE; CORBEIL, 2010).

Após uma extensão em uma direção da *seed*, a mesma é extendida na direção contrária com o complemento-reverso do caminho da própria *seed*. Quando não for mais possível estender nenhuma das *seeds*, os caminhos que se sobrepuserem são unidos, e os contíguos finais são retornados.

Os genomas utilizados nos experimentos do artigo da ferramenta foram das bactérias *Escherichia coli* K-12 MG1655, *Acinetobacter baylyi* ADP1 e *Cryptobacterium curtum* DSM 15641, com tamanho das leituras entre 40 e 50 bp, $k = 21$ e $c = 2$.

3.3.3 HipMer

(GEORGANAS et al., 2014) apresenta diversas técnicas de paralelização de fases da montagem de genoma do *Meraculous* (CHAPMAN et al., 2011), um assembler que combina abordagens baseadas em grafo De Bruijn e *overlap-layout-consensus*. Dentre esas fases, estão análise de k-mer, construção e travessia do grafo De Bruijn em paralelo. Em (GEORGANAS et al., 2015), é apresentado o HipMer, onde diversas otimizações são realizadas nesses algoritmos, e a etapa final de *scaffolding* é paralelizada.

O algoritmo de análise de k-mers em paralelo é executado em três passos. No início de cada um dos passos, cada processador lê a mesma quantidade de dados da sequência, de forma a balancear a carga em termos de k-mers distintos designados a cada processador, que os armazenam os mesmos em tabelas *hash* de forma local. Entretanto, durante a leitura, k-mers errôneos podem ser gerados devido a erros na sequência provenientes do seu sequenciamento, o que aumenta a quantidade de memória necessária para armazenamento de k-mers.

De modo a amenizar esses custos, esse algoritmo utiliza filtro de Bloom. Se um k-mer não foi visto antes, o filtro pode acidentalmente apontá-lo como visto. Entretanto, se um k-mer foi previamente inserido, o filtro certamente reportá-lo como visto, dessa forma, nenhum k-mer

real deixará de ser visto.

No primeiro passo do algoritmo, cada processador transforma leituras em *k*-mers, e realiza uma estimativa local de quantos *k*-mers distintos foram armazenados. Posteriormente, um *reduce* global das estimativas locais é feito, e um filtro de Bloom de tamanho fixo é alocado para cada processador. Esse tamanho é dado pela estimativa obtida pelo *reduce* divido pela quantidade de processadores.

No segundo passo, à medida que cada processo lê uma porção de leituras, uma função determinística mapeia cada *k*-mer a um processador (atribui um *owner* a cada um dos *k*-mers). Todas as ocorrências de um *k*-mer em qualquer um dos processadores são então enviadas a um mesmo processador. Para todo *k*-mer que chega a um processador, se o filtro de Bloom diz que ele foi visto, o mesmo é armazenado em uma estrutura que não permite duplicatas, onde a contagem de frequência é realizada.

No último passo, cada processador monta uma tabela *hash*, com os *k*-mers como chave e suas duas extensões como valor. Ao final do processamento, apenas UU *k*-mers são mantidos, ou seja, apenas *k*-mers que possuem apenas uma extensão de alta qualidade em cada uma das suas extremidades.

A implementação do algoritmo de construção do grafo De Bruijn em paralelo utiliza a representação do *Meraculous*, que gera e percorre um subgrafo linear do grafo De Bruijn completo. Nele, cada nó é um elemento de uma tabela *hash* distribuída, tendo o *k*-mer como chave e um código de duas letras como valor, representando as extensões de cada um dos lados do *k*-mer. O algoritmo consiste em cada um dos n processadores lerem m/n *k*-mers armazenados localmente, sendo m o número de *k*-mers, realizar *hash* da chave e armazenar na região apropriada da tabela distribuída. Para evitar acessos concorrentes à memória, apenas o processo *owner* efetua todas as inserções na tabela.

O algoritmo para travessia do grafo De Bruijn realiza uma travessia em paralelo sem conflitos. De modo a formar um contíguo, cada processador escolhe um *k*-mer aleatório da tabela *hash* distribuída como *seed* e cria um novo subcontíguo contendo somente o *k*-mer escolhido. O processador tenta então estender o subcontíguo em ambas as suas extremidades utilizando as extensões armazenadas como valor na tabela, até que não haja possibilidade de extensão em nenhuma das duas extremidades. Um contíguo então é formado e adequadamente armazenado.

Cada um dos *k*-mers na tabela *hash* distribuída possui uma flag binária com valores *USED* e *UNUSED*. Se o seu valor é *UNUSED*, significa que nenhum outro processador chegou até aquele *k*-mer, e ele está disponível para ser utilizado. Se for *USED*, outro processador já adici-

onou aquele *k*-mer em outro subcontíguo.

Esse algoritmo possui um esquema que sincroniza a comunicação entre os processadores (Figura 3.4). Cada subcontíguo não iniciado se encontra no estado *INACTIVE*. Quando um processador seleciona um *k*-mer para iniciar um contíguo, a sua estrutura é criada o mesmo passa para o estado *ACTIVE*. A seguir, o processador tenta estender o contíguo em ambas as direções. Sempre que achar *k*-mers *UNUSED* em qualquer direção do subcontíguo, adiciona a extensão à esquerda ou à direita do *k*-mer ao subcontíguo e marca os *k*-mers visitados como *USED*. O contíguo permanece no estado *ACTIVE*.

Se o processador não consegue estender um contíguo em nenhuma das direções, ele tenta obter as travas de todos os subcontíguos vizinhos, inclusive o seu próprio, na ordem dos identificadores de cada um dos processadores, de forma a evitar *deadlocks*. Após obter as travas, se os subcontíguos vizinhos estão ambos no estado *ACTIVE*, muda o seu subcontíguo para *ABORTED*, libera as travas em ordem inversa da ordem em que foram obtidas, e inicia outro subcontíguo; se pelo menos um dos subcontíguos vizinhos está no estado *ABORTED*, então o processador une esse subcontíguo ao seu e altera o estado do mesmo para *ACTIVE*. As travas são então liberadas em ordem inversa à obtida e o processador continua a estender o subcontíguo.

Quando o processador atinge ambas as extremidades do contíguo, o coloca em estado de *COMPLETED* e o armazena. Em seguida escolhe outro *k*-mer da tabela *hash* distribuída e inicia outro contíguo. Quando todos os *k*-mers da tabela forem visitados a travessia do grafo De Bruijn estará completa.

A etapa de análise de *k*-mers foi implementada usando C++ com MPI, e integrada com o resto da aplicação, implementada em Berkeley Unified Parallel C (UPC). Utiliza arquitetura de memória distribuída, mas pode ser executada em arquiteturas shared-memory sem modificações.

Experimentos foram conduzidos pelos autores em um supercomputador Cray XC30, que possui um pico de desempenho de 2,57 petaflops/sec, com 5.576 nós, cada um equipado com 64 GB de RAM e dois processadores Intel Ivy Bridge 2,4 GHz de 12 cores, totalizando 133.824 cores, interconectados com a rede Cray Aries utilizando uma topologia Dragonfly.

Também foram realizados experimentos com uma arquitetura *shared-memory*, utilizando um nó do sistema NERSC Carver de 1 TB, contém 4 processadores octa-core Intel Nehalem 2,0 GHz. Consegue *speedup* de 1.94x utilizando *shared-memory* e 6,93x com o Cray XC30.

As leituras dos experimentos variam entre 100 bp e 250 bp, com $k = 51$. Os genomas selecionados foram um genoma humano (NA12878), o genoma hexaplóide do trigo (*Triticum*

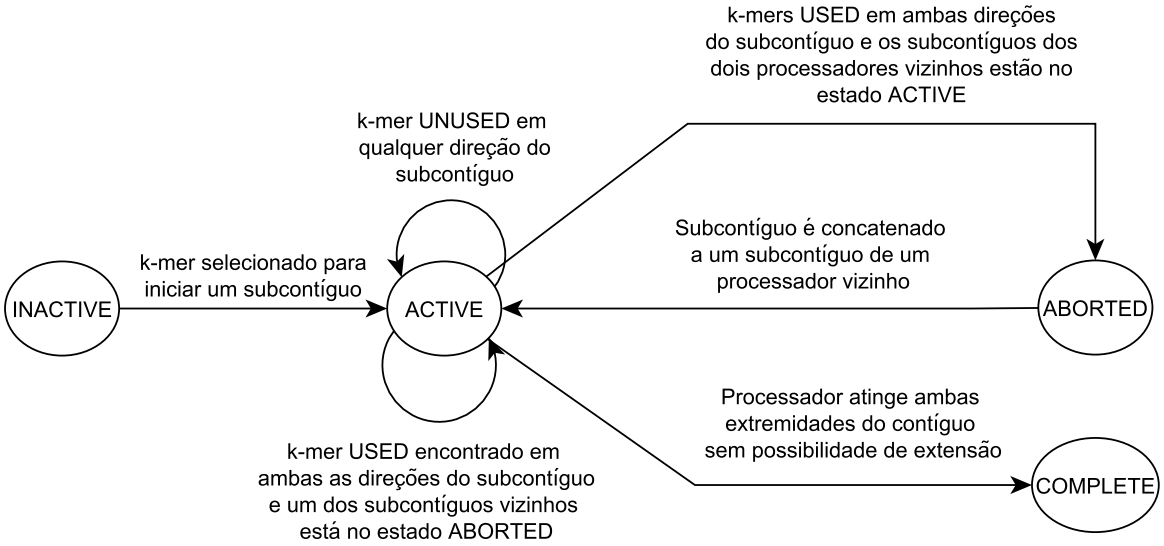


Figura 3.4: Máquina de estado simplificada para sincronização dos processadores na travessia do grafo De Bruijn em paralelo

Adaptado de: (GEORGANAS et al., 2014)

aestivum L.) para montagem no Cray XC30, e a bactéria *E. coli* K-12 MG1655 para a arquitetura *shared-memory*.

3.3.4 A5-miseq

(COIL; JOSPIN; DARLING, 2014) apresenta o A5-miseq, um *pipeline* que inclui diversas ferramentas de bioinformática para efetuar a montagem de genomas sequenciados pela tecnologia Illumina MiSeq. As etapas do *pipeline* (Figura 3.5) são as mesmas de seu antecessor, o *pipeline* A5 (TRITT et al., 2012), que incluem: pré-processamento das leituras (remoção de erros e regiões de baixa cobertura); geração de contíguos; *scaffolding* inicial; correção de erros e *scaffolding* final.

Na etapa de pré-processamento, o *pipeline* Trimmonatic (BOLGER; LOHSE; USADEL, 2014) é utilizado para filtrar regiões de baixa qualidade ou cobertura, e remover sequências adaptadoras (pequenas regiões de DNA sintético presente nos fins das leituras, resultantes do processo de sequenciamento).

Erros são corrigidos pelo SGA (SIMPSON; DURBIN, 2012), que usa um algoritmo de correção baseado nas frequências dos *k*-mers. A ferramenta utiliza uma estrutura de dados chamada de grafo de cadeias de montagem (*assembly string graph*) para representar um conjunto de leituras livres de erros, construído por meio do índice FM (FERRAGINA; GAGIE; MANZINI, 2012).

Esse grafo é um refinamento do grafo de sobreposição, no qual cada leitura é um vértice do grafo, e uma aresta liga dois vértices se houver sobreposição entre as leituras correspondentes. No grafo de cadeias, leituras que estão contidas em outras (i.e. são subcadeias de outras leituras) são consideradas redundantes e não fazem parte do conjunto de vértices do grafo. As arestas são bidirecionais, e possuem como rótulos as subcadeias que fazem parte da sobreposição entre as leituras dos vértices.

O índice FM é uma estrutura de dados desenvolvida para buscas em uma representação comprimida de um texto. Para construir esse índice a partir de um conjunto de leituras, deve-se computar o vetor de sufixos (vetor ordenado de todos os sufixos de uma sequência) para cada uma das leituras. Como uma grande quantidade de memória seria necessária para armazenar genomas maiores, foi implementado um algoritmo distribuído para construir esse índice a partir de cada cadeia do conjunto de leituras.

O algoritmo do SGA percorre cada uma das leituras procurando por bases que não aparecem em um k-mer pelo menos c vezes. A base incorreta do k-mer mais à esquerda da leitura é então alterada para as outras três possíveis. Se alguma delas gera um k-mer com frequência maior que c, então a mudança é mantida. Caso contrário, o mesmo teste é feito com a base incorreta do k-mer mais à direita. Se nenhuma correção for possível, o procedimento se encerra, e a leitura original é retornada. Se nenhuma base incorreta for encontrada na leitura após as correções, ela é retornada com as modificações.

A geração de contíguos é feita pelo IDBA-UD (PENG et al., 2012), um algoritmo de montagem baseado em grafo De Bruijn. Ao invés de usar um valor de k, o algoritmo itera de um valor k_{min} a um valor k_{max}. Em cada iteração, um grafo De Bruijn acumulado é construído a partir das leituras, e os contíguos gerados em cada iteração são usados como leituras da iteração seguinte.

As duas etapas de *scaffolding* são realizadas pelo SSPACE (BOETZER et al., 2011), um programa específico para estender contíguos e gerar *scaffolds* a partir de qualquer conjunto de sequências e de leituras geradas por sequenciadores Illumina. O algoritmo do SSPACE pode ser resumido em três etapas: realizar alinhamento entre as leituras e os contíguos, estendendo cada alinhamento local; calcular informação de *linkage* dos contíguos, para eliminar regiões ambíguas da sequência e identificar *repeats*; concatenar os contíguos lineares, posicionar os *repeats* de acordo com *linkage* e em seguida tentar concatenar novamente.

O próprio A5-miseq se encarrega de avaliar as montagens geradas e procurar eventuais erros. As leituras são alinhadas novamente com os *scaffolds*, aplicando-se técnicas de agrupamento espacial bidimensional para identificar *clusters* de leituras incompatíveis, e posterior-

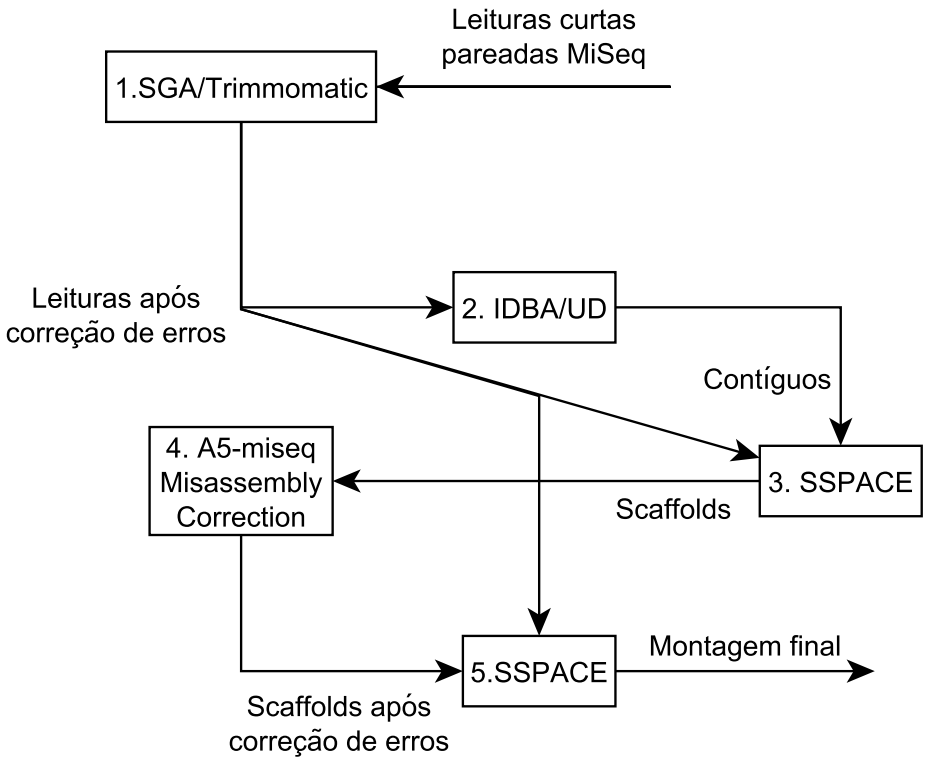


Figura 3.5: Etapas do A5-miseq, com base no pipeline A5
Adaptado de: (TRITT et al., 2012)

mente removendo-as.

Os experimentos do artigo foram realizados com quatro organismos bacterianos: *Bacillus cereus* ATCC 10987, *Rhodobacter sphaeroides* 6G-0125-R, *Mycobacterium abscessus* 2.4.1 e *Vibrio cholerae* CP1032(5), com valor de $k = 23$ para a *R. sphaeroides* e $k = 65$ para as demais e tamanho de leitura 250 bp.

3.4 Montadores de leituras longas

3.4.1 HGAP

(CHIN et al., 2013) é um *workflow* desenvolvido para trabalhar com leituras sequenciadas pela tecnologia PacBio, fazendo uso de uma abordagem *Overlap-Layout-Consensus*. O *workflow* consiste de quatro etapas (Figura 3.6): mapeamento, pré-montagem, montagem e consenso. Inicialmente as leituras de maior tamanho são selecionadas como *seeds*. Em seguida, todas as outras leituras são mapeadas com as *seeds*, gerando leituras pré-montadas.

Na etapa seguinte, é realizada a montagem das leituras mapeadas na etapa anterior por meio

do algoritmo PBDAG-Con, baseado em grafos acíclicos direcionados para representar múltiplos alinhamentos de sequências. Esse algoritmo usa a informação de alinhamento gerada na etapa anterior para alinhar as leituras em uma sequência (grafo) base.

Programação dinâmica é usada sobre o grafo para encontrar a sequência ótima, que pode ser iterativamente usada como sequência base para melhor a qualidade da mesma. O grafo de alinhamento de sequência possui como vértices as bases A, C, T ou G, e como arestas, a conexão com as bases em sequência nas leituras. Cada leitura é um caminho único no grafo, e cada nó e aresta possui um conjunto de leituras que passam por eles.

As leituras pré-montadas podem servir como entrada para qualquer montador que faça uso de abordagem OLC, e tenha capacidade de manusear leituras longas, como o Celera (MYERS et al., 2000). É importante mencionar que o resultado pode apresentar contíguos que ainda possam ser conectados.

A sequência final é então gerada pelo algoritmo *Quiver* na fase de consenso. O *Quiver* faz uso de um algoritmo guloso para maximizar a função de verossimilhança $Pr(R|T)$, sendo $R = \{R_1, R_2, ..., R_k\}$ um conjunto de leituras e T uma sequência base. Primeiramente, as leituras são mapeadas às suas regiões genômicas na sequência. Em seguida, uma sequência base candidata T_s é gerada, a partir de um algoritmo de alinhamento de ordem parcial. Todas as substituições de bases em T_s são efetuadas, e aquelas que aumentarem a verossimilhança são mantidas. Essa operação é repetida até que não seja mais possível maximizar a função.

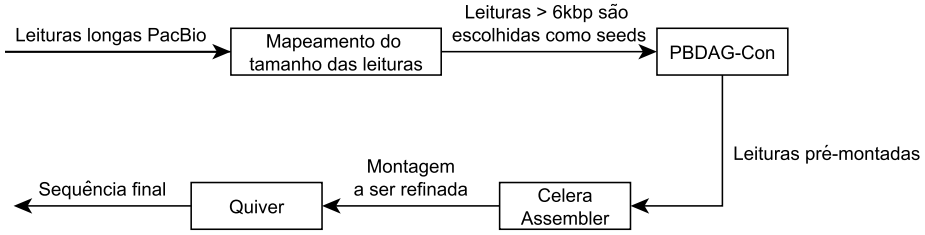


Figura 3.6: Etapas do *workflow* HGAP
Adaptado de: (CHIN et al., 2013)

Foram utilizados nos experimentos as bactérias *Escherichia coli* MG1655, *Meiothermus ruber* DSM 1279 e *Pedobacter heparinus* DSM 2366. O tamanho das leituras para as três bactérias variam entre 7.212 e 18.387 bp.

3.4.2 Canu

(KOREN et al., 2017) é um *pipeline* derivado do Celera (MYERS et al., 2000), desenvolvido para trabalhar com leituras sequenciadas por tecnologia PacBio ou Nanopore. No início de cada uma das etapas (correção, poda e montagem), é construído e armazenado um índice das leituras, um histograma dos k-mers e um índice de todas as sobreposições possíveis entre dois pares de leituras. Para identificar as sobreposições, a primeira etapa utiliza o algoritmo MHAP, enquanto as outras duas usam o overlapInCore (MYERS et al., 2000).

O algoritmo MHAP constroi um filtro de sobreposição em duas etapas. A primeira etapa tenta identificar pares de leituras que potencialmente possuem uma sobreposição, por meio de um peso TF-IDF (frequência do termo-inverso da frequência nos documentos), onde o termo corresponde a um k-mer e o documento a uma determinada leitura. A segunda etapa tenta estimar o tamanho e a taxa de erro das sobreposições, verificando funções de distância e similaridade entre conjuntos de k-mers.

A correção de erros é feita a partir da informação de sobreposição entre as leituras. Um filtro global e um filtro local são usados para determinar quais sobreposições vão ser usadas para corrigir cada leitura. O filtro global atribui um *score* para cada sobreposição e escolhe as C melhores para cada leitura.

Em seguida, após as correções serem efeituadas, o filtro local escolhe as $2C$ melhores sobreposições para a correção. Uma correção é feita gerando-se um grafo acíclico direcionado a partir das dos alinhamentos entre as sequências, e o maior caminho (de maior peso) é percorrido para produzir a sequência corrigida.

Na etapa de poda, é calculada a sobreposição entre as leituras corrigidas, e as regiões que não se alinham com nenhuma outra leitura são removidas. Em outras palavras, é removida toda e qualquer região que não for coberta por pelo menos C sobreposições com taxa de erro de no máximo E e seja de tamanho menor que L.

A etapa de montagem faz uso de uma adaptação do OLC, chamada de grafo de melhor sobreposição. Uma melhor sobreposição é a sobreposição de maior tamanho que envolva somente as terminações (3' e 5') de ambas as leituras. Leituras com alta taxa de erro de sobreposição e alta diferença de tamanho de sobreposição são removidas, enquanto as leituras restantes, juntamente com as melhores sobreposições, formam o grafo de melhor sobreposição. A montagem é gerada de forma similar a (CHIN et al., 2013), com o algoritmo PBDAG-Con.

Foram usados nos experimentos dados PacBio dos seguinte organismos: *Escherichia coli* (bactéria), *Arabidopsis thaliana* (angiosperma), *Caenorhabditis elegans* (verme), *Drosophila*

melanogaster (mosca) e genoma humano (CHM1). Também foram usados dados Nanopore das bactérias *Escherichia coli*, *Bacillus anthracis* e *Yersinia pestis*.

3.5 Montadores híbridos

3.5.1 SPAdes

(ANTIPOV et al., 2015) apresenta o algoritmo HybridSPAdes, que realiza montagem a partir de leituras curtas e longas. Esse algoritmo foi incorporado ao montador SPAdes (BANKEVICH et al., 2012), possibilitando montagem híbrida com tecnologia Illumina/IonTorrent e PacBio/Nanopore. Inicialmente o HybridSPAdes constrói um *assembly graph* a partir das leituras curtas, fazendo uso do próprio SPAdes. Esse grafo consiste de um grafo De Bruijn simplificado após remoção de erros.

Em seguida, as leituras longas são mapeadas ao grafo, onde cada caminho corresponde a como cada uma dessas leituras percorre esse grafo. O HybridSPAdes usa um valor t (por padrão $t = 13$) para determinar *seeds*, onde cada leitura é transformada em um conjunto de t-mers. Um mapeamento ocorre se uma leitura e uma aresta no grafo compartilham um mesmo t-mer. Um caminho passa por determinada aresta se pelo menos 8 t-mers da leitura original forem mapeados àquela aresta.

Com a inserção de leituras longas no grafo, a cobertura total dos k-mers pode diminuir, afetando k-mers de regiões isoladas da sequência, que são representados por apenas uma aresta, um *coverage gap*. Essa aresta pode ser transformada em outras duas: *sink*, que tem como destino um vértice com grau de saída zero, e *source*, que tem como origem um vértice com grau de entrada zero.

Se uma leitura possui mapeamento com uma aresta *sink* e uma aresta *source*, essa leitura pode ser selecionada para corrigir os *gaps*. Todas as leituras que mapeam com o mesmo par de arestas *sink* e *source* são selecionados para solucionar o *gap* com a sequência de consenso gerada a partir dessas leituras.

Na etapa final, o conjunto de caminhos que passam por pelo menos duas arestas longas no grafo é convertido em contíguos. Esses caminhos são expandidos de forma iterativa, pelo *framework* ExSPAnder (PRJIBELSKI et al., 2014). Dado um caminho, o ExSPAnder tenta extendê-lo com uma aresta que tem origem em um de seus vértices folha, selecionada de acordo com uma regra.

Os experimentos foram efetuados com diversos *datasets* de leituras curtas e longas das

bactérias *Escherichia coli* K-12, *Meiothermus ruber*, *Streptomyces sp.* PAMC 26508 e da bactéria candidata TM6 (TM6SC1). Foram utilizadas leituras curtas de 100 e 150 bp e leituras longas de tamanho entre 1.410 e 10.598 bp.

3.5.2 DBG2OLC

(YE et al., 2016) apresenta uma abordagem de montagem híbrida que trabalha com ambos os paradigmas de grafo De Bruijn (DBG) quanto grafo de sobreposição (OLC). O algoritmo (Figura 3.7) inicia com a construção do grafo De Bruijn, a partir das leituras curtas, e a geração de contíguos a partir do mesmo. Em seguida, esses contíguos são mapeados com as leituras longas.

Cada leitura longa é comprimida em um conjunto de identificadores de contíguos nesse mapeamento. Um identificador é gerado se o número de *k*-mers distintos compartilhados entre determinado contíguo e uma leitura é maior que um *threshold* determinado com base no tamanho do contíguo. Além dos identificadores, a orientação do contíguo no mapeamento também é armazenada.

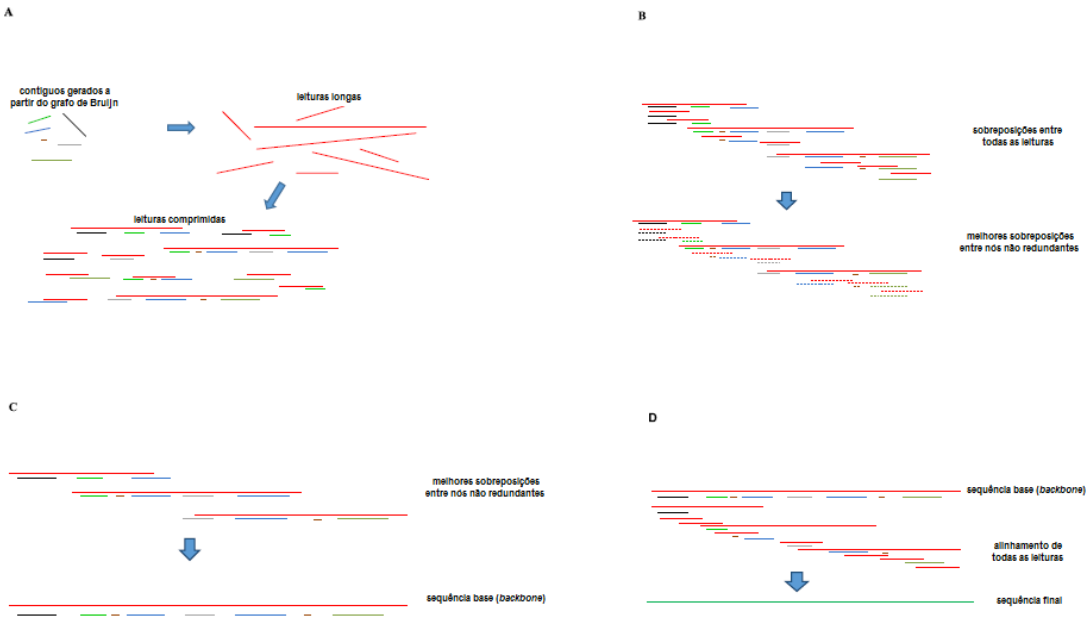


Figura 3.7: Etapas do DBG2OLC
Adaptado de: (YE et al., 2016)

As leituras são então submetidas a um procedimento de alinhamento múltiplo de sequências para remover erros estruturais. Nesse procedimento, um índice invertido é construído a partir dos identificadores, que seleciona as leituras candidatas ao alinhamento baseado nos identificadores que compartilham entre si. Os *scores* de alinhamento são calculados pelo algoritimo

de Smith-Waterman, levando em consideração, similarmente à etapa anterior, o tamanho dos contíguos e os *k*-mers compartilhados entre eles.

A etapa seguinte consiste em criar um grafo de sobreposição a partir das leituras longas após a compressão e remoção de erros. Nesse grafo, cada nó representa uma leitura comprimida. Nós redundantes são primeiro removidos de acordo com a informação dos identificadores.

Para cada um dos nós restantes, o respectivo nó anterior e nó seguinte são localizados de acordo com o *score* de alinhamento, e são ligados com o auxílio das leituras curtas, produzindo uma sequência que será utilizada como base (*backbone*) no alinhamento final. Na etapa final, todas as leituras são alinhadas com a sequência base, onde as regiões lineares no grafo que não possuirem ramificações geram as sequências.

Foram realizados experimentos com genomas de organismos de vários tamanhos: trigo *Saccharomyces cerevisiae* (pequena amostra de 12 Mbp), humano *Homo sapiens* (3 Gbp), angiosperma *Arabidopsis thaliana* (120 Mbp) e bacteriano *Escherichia coli*. O tamanho médio das leituras era de aproximadamente 14.500 bp, e valor $k = 17$ para a etapa inicial de construção do grafo De Bruijn.

3.6 Meta-montagem

O avanço das tecnologias de sequenciamento tem permitido o surgimento de novas ferramentas e algoritmos para realizar tal tarefa. Devido à grande disponbilidade dessas tecnologias, uma nova abordagem de montagem de genomas foi desenvolvida, chamada híbrida (POP, 2009).

A montagem híbrida faz uso de leituras provenientes de diferentes sequenciadores para reconstruir o genoma, em sua maioria, utilizando métodos baseados em *overlap-layout-consensus*. Essa abordagem também pode fazer uso de múltiplos montadores, devido ao fato de que dados gerados por sequenciadores distintos possuem suas próprias características.

Ao invés de se realizar a montagem a partir de leituras, a estratégia híbrida também pode usar montagens geradas por *assemblers*. Essa técnica, chamada de meta-montagem, combina os resultados (contíguos e/ou *scaffolds*) produzidos por diferentes ferramentas de montagem para produzir uma nova sequência.

Não se deve confundir no entanto, os conceitos de montador híbrido e de montagem híbrida. Em se tratando de montadores, ”híbrido”se refere à capacidade de um montador trabalhar com leituras curtas e longas. A respeito do processo de montagem, ”híbrida”se refere a um

processo de montagem que pode fazer uso de mais de um tipo de estratégia de montagem (DBG/OLC), sequenciador (independente do tipo de leitura) ou conjunto de dados de entrada (leituras/montagens).

3.6.1 GAM-NGS

(VICEDOMINI et al., 2013) é uma ferramenta que combina duas montagens sem a necessidade de alinhamento global entre os contíguos, identificando fragmentos similares entre as duas montagens, i.e. regiões que compartilham uma grande quantidade de leituras. Além das montagens, o GAM-NGS necessita de um arquivo contendo alinhamentos para cada conjunto de leituras e cada montagem.

A etapa inicial consiste em contruir blocos entre as duas montagens. Dadas duas montagens *M* e *S*, as leituras que são mapeadas em *M* são inicialmente armazenadas em uma tabela *hash*. Em seguida, cada leitura mapeada em *S* é analisada: se ela não estiver na tabela *hash*, não será usada para construir blocos; se for adjacente a algum bloco existente, o bloco é estendido com essa leitura. Caso contrário, um bloco é construido com a leitura.

Na etapa seguinte, é construído um grafo de contíguos, onde os vértices representam os contíguos presentes em *M* e *S*. Uma aresta conecta dois vértices se eles pertecem a diferentes montagens e são cobertos por pelo menos um bloco. Cada aresta receberá um peso de acordo com os blocos compartilhados entre os contíguos de seus vértices, enquanto o peso dos vértices é calculado pela média dos pesos das arestas que incidem no mesmo.

Após o grafo de contíguos estabelecer a ordem dos blocos entre os contíguos de ambas as montagens, é gerado um grafo de montagens, no qual um nó é criado para cada bloco, ao passo que arestas ligam blocos que compartilham ao menos um quadro (sequência de leituras adjacentes que mapeam uma mesma montagem) no mesmo contíguo. Cada caminho no grafo representa uma sequência de blocos, no qual dois blocos são consecutivos em pelo menos uma montagem.

Posteriormente à remoção de erros no grafo, é executado um algoritmo de alinhamento semi-global sobre o mesmo (UKKONEN, 1985), de forma que contíguos com alto grau de similaridade sejam combinados. Uma característica do GAM-NGS é não gerar sequências de consenso. Ao invés disso, a saída é formada pelas sequências pertencentes às montagens que localmente apresentam as melhores estatísticas, juntamente com contíguos que não fizeram parte de nenhuma junção.

3.6.2 GARM

(SOTO-JIMENEZ; ESTRADA; SANCHEZ-FLORES, 2014) apresenta o GARM, um *pipeline* que combina montagens de diferentes *assemblers*, independentemente de qual tecnologia de sequenciamento foi utilizada. O *pipeline* necessita como entrada somente as duas montagens a serem combinadas, conseguindo combinar montagens que contém apenas contíguos, apenas *scaffolds* ou ambos, como mostrado na Figura 3.8

Inicialmente, o GARM calcula diversas estatísticas de cada montagem, como número e tamanho médio dos fragmentos, e baseado nas mesmas escolhe a montagem de melhor resultado como base (ou seja, os alinhamentos serão feitos sobre ela). Um passo adicional é realizado se existem *scaffolds* nos dados de entrada, onde os mesmos são armazenados e utilizados posteriormente. Em seguida, são divididos em contíguos e as estatísticas são calculadas.

A detecção de sobreposições entre contíguos é então realizada pelo alinhador NUCmer (DELCHER et al., 2002), podendo ser executada de forma distribuída se houverem múltiplos *cores* disponíveis. O NUCmer faz uso de uma abordagem OLC que remove sobreposições de contíguos de diferentes montagens e seleciona sequências que já foram alinhadas e estão dentro de outras sequências.

Na primeira etapa, o NUCmer gera um mapeamento de todas as posições dos contíguos em cada uma das leituras presentes na entrada. Logo após, faz uso do MUMmer (DELCHER et al., 2002) para encontrar todos os alinhamentos em cada uma das montagens, e os mesmos são mapeados de volta aos contíguos. Na etapa seguinte, os alinhamentos são agrupados juntamente com os contíguos, de acordo com uma distância fornecida pelo usuário. As sequencias são então alinhadas por meio de uma versão modificada do algoritmo de Smith-Waterman.

O GARM usa as informações acerca dos *scaffolds*, se presentes, para inserir os contíguos que não foram combinados e faziam parte de um *scaffold* de volta no mesmo. No final, gera um arquivo com os contíguos que foram combinados e um arquivo contendo as sequências que não possuem sobreposição entre si, além dos contíguos que foram filtrados pelo NUCmer. Um arquivo contendo os *scaffolds* finais também é gerado se as montagens iniciais continham *scaffolds*.

3.6.3 Metassembler

(WENCES; SCHATZ, 2015) apresenta o Metassembler, um algoritmo capaz de combinar diversas montagens por meio de sucessivas junções par a par, a partir de leituras, montagens,

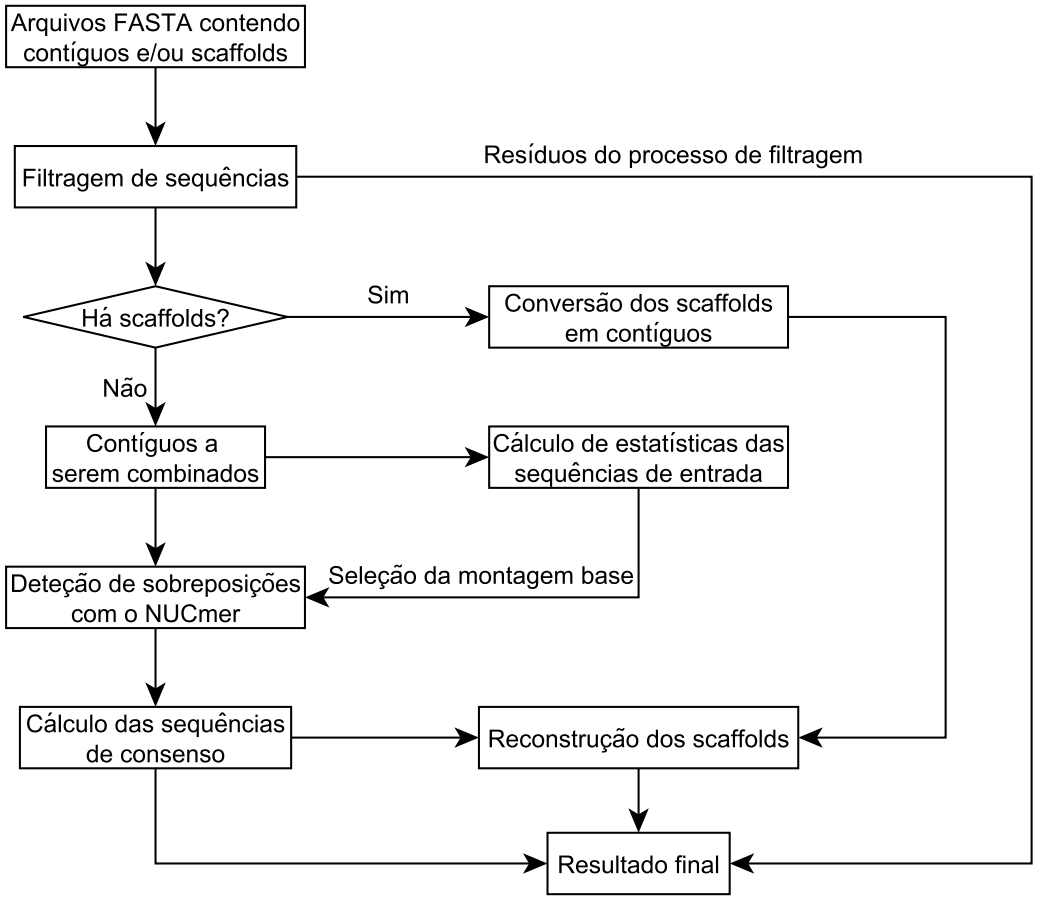


Figura 3.8: Fluxograma do GARM
Adaptado de: (SOTO-JIMENEZ; ESTRADA; SANCHEZ-FLORES, 2014)

e *jump libraries* (ferramentas para auxiliar o mapeamento e sequenciamento de regiões cromossômicas). O alinhamento par a par é realizado na ordem informada pelo usuário ou por alguma métrica, sem qualquer restrição de como as montagens foram geradas.

Assim como o GARM, é utilizado o NUCmer para efetuar alinhamento entre as duas sequências iniciais, e o resultado é usado para mapear cada região da sequência principal com a melhor região correspondente na secundária, de acordo com uma função de maximização calculada pelo produto entre o tamanho e a identidade do alinhamento (porcentagem de bases idênticas nas mesmas posições).

Em seguida, o Metassembler mapeia as *jump libraries* fornecidas juntamente com as montagens, por meio do algoritmo Bowtie2 (LANGMEAD; SALZBERG, 2012), que usa a estatística CE (ZIMIN et al., 2008) para indicar a possibilidade de se expandir ou comprimir determinada região de uma sequência, o que pode ajudar a definir qual montagem tem maior grau de confiança em casos de conflitos entre alinhamentos.

A etapa seguinte consiste em comparar e unir as duas sequências. Todas as regiões alinhadas da sequência principal são adicionadas à sequência de consenso, ao passo que a estatística CE é usada para determinar qual sequência fará parte do consenso em regiões não alinhadas:

- Quando detectada inserção em uma sequência e o valor CE dessa região for positivo acima de um *threshold* (geralmente 3), a inserção é rejeitada e a outra sequência é selecionada. Caso contrário, é escolhida a própria inserção.
- Quando as duas sequências possuem inserções não alinhadas, o tamanho e o valor CE delas irão determinar qual das inserções serão escolhidas. Inserções que possuem menor tamanho e valor CE negativo abaixo do *threshold* são descartadas.

Se mais de duas montagens são fornecidas como entrada, após essa etapa, o procedimento é novamente repetido, dessa vez com a montagem de consenso gerada e a melhor montagem fornecida na entrada, determinada pelo usuário ou de acordo com alguma estatística.

3.7 Considerações finais

Neste capítulo foi discutido sobre o processo de montagem de genomas, ferramentas e suas classificações, além de apresentar diversos montadores, agrupados com base no critério de tamanho das leituras usadas: curtas, longas ou ambas (híbrido).

Dentre os montadores de leituras curtas, os seguintes foram apresentados: ABySS, um dos primeiros montadores baseados em grafo De Bruijn que possui implementação distribuída; Ray, que combina DBG e algoritmos gulosos, ao utilizar heurísticas para percorrer o grafo, invés de caminhos eulerianos; HipMer, uma série de algoritmos otimizados para obter escalabilidade, capaz de trabalhar com genomas grandes; e por fim o A5-miseq, um *pipeline* que combina outras ferramentas de montagem.

Sobre os montadores longos, se destacam o HGAP e o Canu. O HGAP foi desenvolvido para trabalhar exclusivamente com leituras PacBio, e faz parte do pacote de análise de sequências fornecido pela Pacific Biosciences, ao passo que o Canu foi desenvolvido a partir do *pipeline* Celera, tendo o seu enfoque em leituras longas, sequenciadas a partir de tecnologia PacBio ou Nanopore.

Acerca dos montadores híbridos, os principais são o SPAdes e o DBG2OLC. O SPAdes foi desenvolvido como um montador curto, e posteriormente foi adicionado o algoritmo HybridSPAdes, que efetua montagem híbrida. O DBG2OLC, como o nome sugere, faz uso dessas duas

técnicas de grafo para gerar sequências de forma híbrida, realizando montagem curta com DBG e mapeando leituras ao resultado com OLC.

Também foi apresentado o conceito de meta-montagem, onde montagens são produzidas a partir de outras, e não a partir de leituras, como é feito pelos montadores convencionais. Ferramentas que fazem uso desse conceito incluem: GAM-NGS, que não realiza alinhamento global e não gera sequência de consenso; GARM, combina montagens sem precisar de informação adicional; e Metassembler, capaz de combinar mais de duas montagns por execução.

No capítulo seguinte é discorrido a respeito da ferramenta proposta, desde o seu projeto inicial, até a forma presente do *pipeline*, bem como o estudo de caso e as ferramentas que influenciaram o seu desenvolvimento.

Capítulo 4

PROPOSTA DO RESHAPE: UM PIPELINE HÍBRIDO PARA MONTAGEM DE GENOMAS BACTERIANOS

4.1 Estudo de caso

A *Pseudomonas aeruginosa* CCBH4851 é uma bactéria encontrada em hospitais brasileiros, como um agente causador de infecções hospitalares. Essa bactéria possui resistência a antibióticos e alto grau de virulência em pacientes internados, o que leva à necessidade de estudos com objetivo de se desenvolver novos tratamentos contra esse organismo (VALLET-GELY; BOCCARD, 2013).

Os dados de sequenciamento da bactéria são de propriedade da Fundação Oswaldo Cruz (Fiocruz) (SILVEIRA et al., 2014). A Fiocruz possui um projeto que tem por objetivo criar o modelo computacional da célula completa da *P. aeruginosa*. Com esse modelo computacional, fenômenos complexos poderão ser previstos, no que diz respeito a moléculas individuais e suas interações.

O desenvolvimento do modelo computacional depende da construção da rede metabólica, onde reações enzimáticas e de transporte associadas a genes previamente catalogados em diferentes bases de dados são recuperados, a partir dos dados genômicos da bactéria. Entretanto, o genoma da *P. aeruginosa* CCBH4851 ainda não foi completamente montado, o que motivou a sua escolha como objeto de estudo desse trabalho.

4.2 Primeira versão: montagem com leituras curtas

Em um primeiro momento, foram realizados diversos experimentos com os dados da *P. aeruginosa*, disponíveis no banco de nucleotídeos do NCBI (número de acesso do projeto de sequenciamento JPSS00000000.1), e fornecidos pela Fiocruz, mostrados na Tabela 4.1. Vale salientar que os experimentos com o A5-miseq foram realizados somente após a elaboração do *workflow* da Figura 4.1.

Tabela 4.1: Experimentos preliminares com as leituras de 300 e 500 bp da *P. aeruginosa* CCBH4851

Montador	ABySS	Ray	HipMer	A5-miseq
Versão	1.9.0	2.3.1	0.9.4	25-08-2016
Fragmentos gerados com leituras de 300 bp	390	400	362	119
Fragmentos gerados com leituras de 500 bp	351	393	308	109

Leituras de 300 e 500 bp, sequenciados pela tecnologia Illumina MiSeq, serviram de entrada para diversos montadores em paralelo. Dentre eles, o que apresentou melhor resultado (ou seja, menor quantidade de fragmentos na montagem final) foi o HipMer, com 308 *scaffolds*, obtido com as leituras de 500 bp.

Em seguida, a montagem disponível no NCBI, que possui 150 contíguos, foi combinada com a melhor montagem gerada pelo HipMer por meio do GARM, escolhido dentre os meta-montadores analisados pelo fato de não necessitar qualquer informação adicional relacionada a um genoma de referência para combinar diferentes montagens. O resultado final gerou 88 fragmentos, dos quais 56 são contíguos e 32 são *scaffolds*, uma redução na quantidade de fragmentos de aproximadamente 40%.

Com base nesses experimentos, foi idealizado um *workflow* (Figura 4.1) para montagem de genomas que combinava diversas ferramentas de novo, e que pudesse ser extendido a outros genomas além da *P.aeruginosa*. A estrutura do *workflow* permitia o uso de apenas uma das ferramentas, se desejado. Alguns passos foram adicionados devido a requerimentos do HipMer.

Posteriormente, novos experimentos foram realizados utilizando o A5-miseq como montador de leituras curtas, ao invés do HipMer. As duas montagens geradas pelo A5-miseq com as leituras de 300 e 500 bp foram combinadas com a montagem disponível no NCBI via GARM, cada. As duas montagens resultantes foram então novamente fornecidas ao GARM para uma segunda execução, onde o resultado final apresentou apenas 35 contíguos, o que deixou claro que o *workflow* precisava ser revisto.

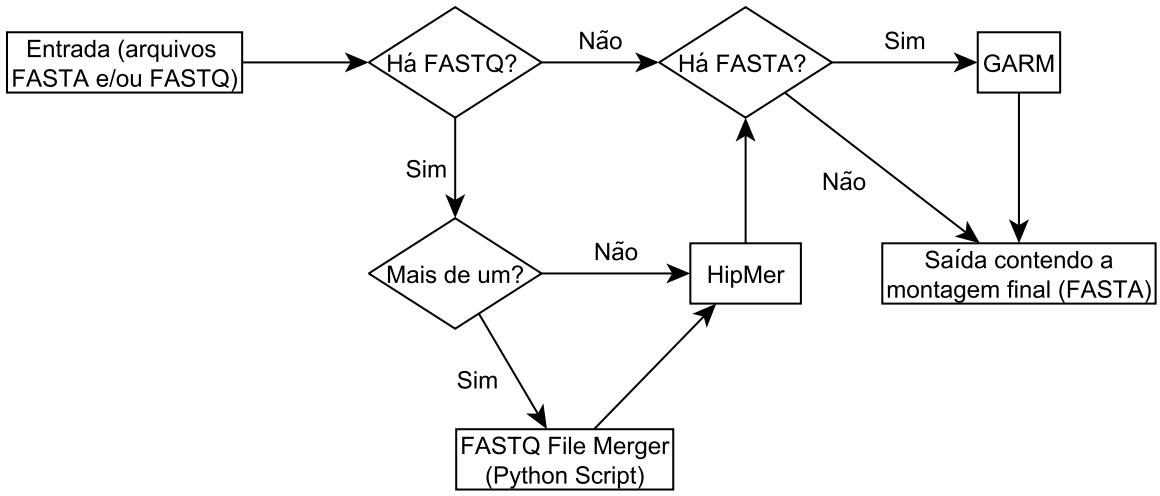


Figura 4.1: *Workflow* inicialmente proposto

4.3 Segunda versão: adição de suporte a leituras longas

Após os experimentos com o A5-miseq, o sequenciamento da *P. aeruginosa* CCBH4851 com tecnologia PacBio RS foi financiado pela Fiocruz. Com o acesso a esses dados, foram investigados métodos que lidam com leituras longas de forma a inserí-las na ferramenta, o que levou aos montadores híbridos, visto que eles constroem montagens a partir de leituras provenientes de diferentes tecnologias de sequenciamento.

Os principais montadores híbridos estudados foram o SPAdes e o DBG2OLC, visto que ambos trabalham simultaneamente com leituras curtas e longas, e dão suporte a leituras sequenciadas com tecnologia Illumina e PacBio. Nesses montadores, a estratégia híbrida consiste em gerar um grafo De Bruijn a partir das leituras curtas e em seguida mapeá-las com as leituras longas, de forma a gerar uma sequência de consenso.

A diferença entre as abordagens dos dois montadores, é que o SPAdes faz o mapeamento das leituras longas em cima do próprio grafo de Bruijn para posteriormente percorrê-lo e gerar as sequências. Ao passo que o DBG2OLC primeiramente realiza uma pré-montagem a partir das leituras curtas, e então alinha as leituras longas com os contíguos gerados, onde as melhores sobreposições geram as sequências.

Inspirado nessas estratégias, optou-se por utilizar o GARM como algoritmo para gerar as sequências de consenso, aproveitando também a sua característica de combinar duas montagens independente de quaisquer tecnologias de sequenciamento e montagem. Dessa forma, foi possível adicionar um comportamento híbrido à ferramenta.

Para tal, era necessário ter à disposição montagens de qualidade, provenientes de montadores de leituras curtas e longas. Em termos de montadores curtos, os experimentos anteriores apontaram que o A5-miseq atendia a esse quesito. A literatura apresenta alguns montadores longos, sendo os principais o Celera, o HGAP e o Canu.

Enquanto o Celera foi descontinuado, o HGAP está presente no conjunto de aplicações mantido pela própria Pacific Biosciences (dona da tecnologia PacBio RS), chamados coletivamente de SMRT Analysis. No entanto, diversas especificações como a necessidade de um navegador *web* e de *clusters* com vários nós para a execução, torna o HGAP difícil para ser integrado a qualquer ferramenta. Portanto, decidiu-se pelo uso do Canu, que foi desenvolvido a partir do Celera.

A Figura 4.2 mostra o projeto inicial do *pipeline*, batizado de reSHAPE (*Sousa's Hybrid Assembly Pipeline*). Nele, A5-miseq e Canu realizam as montagens de leituras curtas e longas, respectivamente. Em seguida, o GARM efetua uma meta-montagem, combinando as duas sequências com uma técnica de *Overlap–Layout–Consensus* (OLC).

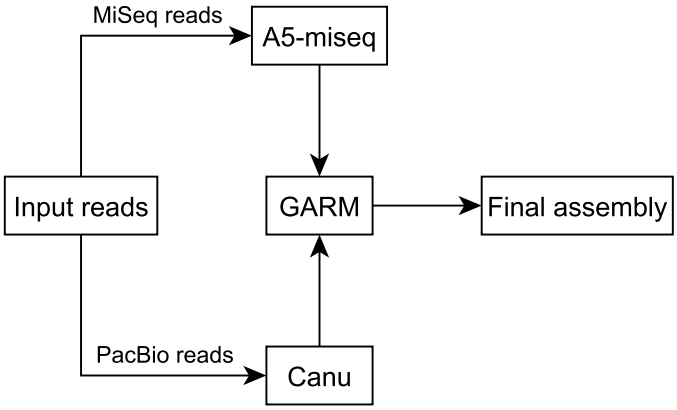


Figura 4.2: Projeto inicial do *pipeline*

4.4 Terceira versão: redução de fragmentos

As montagens geradas pelo A5-miseq eram na maioria das vezes bastante fragmentadas, em comparação com as montagens produzidas pelo Canu, o que afetava o resultado final. Isso motivou a adição de mais uma etapa logo após a execução do A5-miseq, com o propósito de mitigar a fragmentação.

Como mostrado na Figura 4.3, foi dedicido incluir uma execução preliminar do GARM como um estágio adicional de *scaffolding*, de forma a combinar uma sequência previamente

montada (sem qualquer restrição de tecnologia de sequenciamento) do mesmo organismo juntamente com a saída do A5-miseq. Como resultado, o reSHAPE consegue reduzir o número de contíguos ao passo que realiza correção de erros.

A nova etapa permite o uso de montagens de alta qualidade como *templates* para o processo de montagem, proporcionando um maior grau de confiança aos resultados, ao mesmo tempo que aprimora os contíguos resultantes.

Essa característica tornou possível produzir montagens menos fragmentadas, comparadas com outros montadores híbridos, como SPAdes e DBG2OLC, e foi fundamental para melhorar as montagens de diversas bactérias disponíveis no NCBI, diminuindo a quantidade de contíguos, enquanto mantém estatísticas similares às montagens originais. Os resultados das comparações mencionadas serão exibidos no Capítulo 5.

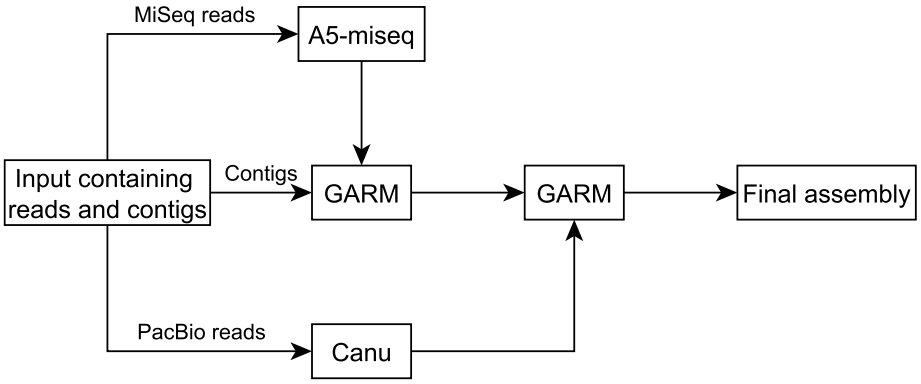


Figura 4.3: Visão geral do reSHAPE

Após o processo de montagem, é executado o QUAST (GUREVICH et al., 2013), uma ferramenta que permite avaliar sequências, fornecendo diversas estatísticas. No reSHAPE, o QUAST é usado para prover informações acerca das montagens geradas em cada uma das quatro etapas anteriores.

O código-fonte do reSHAPE está disponível em: https://github.com/heriosousa/reSHAPE. Informações como requerimentos, instalação e como utilizar a ferramenta podem ser encontradas nesse repositório, bem como mais informações acerca das ferramentas usadas para compor o *pipeline*.

4.5 Considerações finais

No presente capítulo foram apresentadas as etapas de desenvolvimento do *pipeline* híbrido proposto reSHAPE, que combina diversas ferramentas de montagem, assim como as próprias ferramentas que fazem parte do mesmo. A sua atual estrutura teve como inspiração as abordagens utilizadas pelos montadores híbridos apresentados e as análises feitas após a obtenção dos dados da *P. aeruginosa*.

O capítulo a seguir apresenta uma análise do *pipeline*. Em um primeiro momento seus resultados são comparados com as montagens geradas pelo SPAdes e pelo DBG2OLC para a *P. aeruginosa*. Em seguida, essa comparação é efetuada para um conjunto de bactérias com dados disponíveis na base de nucleotídeos NCBI.

Capítulo 5

AVALIAÇÃO DA FERRAMENTA

5.1 Comparação com outras ferramentas híbridas

Finalizado o *pipeline* do reSHAPE, o seu desempenho foi comparado com dois dos principais montadores híbridos da literatura: SPAdes e DBG2OLC. Inicialmente, as três ferramentas foram executadas com a *P. aeruginosa* CCBH4851, e após o reSHAPE apresentar resultados satisfatórios em relação aos outros montadores, decidiu-se por realizar uma nova comparação com dados publicamente disponíveis de outras bactérias.

Para efetuar a comparação, as montagens finais geradas por cada uma das ferramentas foram avaliadas de acordo com as métricas fornecidas pelo QUAST. As próximas seções apresentam os *reports* do QUAST para cada uma das ferramentas em relação a cada uma das bactérias escolhidas. As principais métricas a serem consideradas na avaliação das montagens são:

- # contigs: número total de contíguos
- Largest contig: tamanho do maior contíguo
- Total length: número total de bases
- N50: tamanho de contíguo no qual, usando contíguos de mesmo tamanho ou maior, se produz 50% do número de bases da montagem
- # genes: número estimado de genes encontrados por uma ferramenta de predição de genes

É importante salientar que todas as estatísticas produzidas são baseadas em contíguos de tamanho maior que 500 bp, a menos que o contrário seja indicado.

5.2 Resultados com a *P. aeruginosa* CCBH4851

Como mencionado anteriormente, todas as leituras e montagens usadas nos experimentos mostrados na Tabela 5.1 foram fornecidas pela Fiocruz. Nela, as estatísticas demonstram que a montagem gerada pelo reSHAPE é bem menos fragmentada do que as geradas pelos outros montadores híbridos. Embora o resultado do DBG2OLC apresente um número de contíguos igualmente baixo, o tamanho do maior contíguo, o valor de N50 e a quantidade de genes detectados são menores quando comparados ao do reSHAPE.

Tabela 5.1: Estatísticas das montagens finais geradas para a bactéria *P. aeruginosa* CCBH4851

Assembly	reSHAPE	SPAdes	DBG2OLC
# contigs	**3**	13	4
# contigs (>= 0 bp)	3	879	4
# contigs (>= 1000 bp)	3	10	4
# contigs (>= 5000 bp)	3	7	4
# contigs (>= 10000 bp)	3	6	4
# contigs (>= 25000 bp)	3	6	2
# contigs (>= 50000 bp)	3	6	2
Largest contig	**6419965**	2511154	4294673
Total length	**7325262**	6817664	6819024
Total length (>= 0 bp)	7325262	7024935	6819024
Total length (>= 1000 bp)	7325262	6815957	6819024
Total length (>= 5000 bp)	7325262	6807112	6819024
Total length (>= 10000 bp)	7325262	6801241	6819024
Total length (>= 25000 bp)	7325262	6801241	6787850
Total length (>= 50000 bp)	7325262	6801241	6787850
GC (%)	66.09	66.10	65.44
N50	**6419965**	1169430	4294673
N75	6419965	1005830	2493177
L50	1	2	1
L75	1	4	2
# N's per 100 kbp	0.00	0.01	0.00
# predicted genes (unique)	**6945**	6299	4790
# predicted genes (>= 0 bp)	7350	6310	4790
# predicted genes (>= 300 bp)	6348	5689	3122
# predicted genes (>= 1500 bp)	823	875	387
# predicted genes (>= 3000 bp)	73	88	29

5.2.1 Resultados com outras bactérias

Após os resultados com o estudo de caso, decidiu-se por repetir os experimentos com outras três bactérias, escolhidas devido a disponibilidade de seus dados (leituras MiSeq, PacBio e montagens). As três bactérias selecionadas foram:

- *E. coli* O157:H7 str. F8092B (número de acesso no NCBI: AVCD00000000.1, estatísticas dos experimentos na Tabela 5.2)
- *R. sphaeroides* 2.4.1 (número de acesso no NCBI: AKBU00000000.1, estatísticas dos experimentos na Tabela 5.3)
- *F. tularensis* 99A-2628 (número de acesso no NCBI: AUXB00000000.1, estatísticas dos experimentos na Tabela 5.4)

As leituras Miseq usadas para a *E. coli* O157:H7 str. F8092B estão disponíveis no *European Nucleotide Archive* (ENA): ⟨http://www.ebi.ac.uk/ena/data/view/SRR941218⟩, enquanto as leituras PacBio podem ser encontradas na página da ferramenta PBcR, um *pipeline* para correção de erros em leituras PacBio: ⟨ftp://ftp.cbcb.umd.edu/pub/data/PBcR/closure_paper/BT/filtered_subreads.200X.fastq.bz2⟩.

Tabela 5.2: Estatísticas das montagens finais geradas para a bactéria *E. coli* O157:H7 str. F8092B

Assembly	reSHAPE	SPAdes	DBG2OLC
# contigs	**5**	86	80
# contigs (>= 0 bp)	5	494	80
# contigs (>= 1000 bp)	5	62	80
# contigs (>= 5000 bp)	5	34	78
# contigs (>= 10000 bp)	5	24	74
# contigs (>= 25000 bp)	5	18	45
# contigs (>= 50000 bp)	5	15	34
Largest contig	**4324771**	1221528	382337
Total length	5206587	**5442362**	4524382
Total length (>= 0 bp)	5206587	5510557	4524382
Total length (>= 1000 bp)	5206587	5425741	4524382
Total length (>= 5000 bp)	5206587	5371944	4516161
Total length (>= 10000 bp)	5206587	5304549	4484930
Total length (>= 25000 bp)	5206587	5214460	4063116
Total length (>= 50000 bp)	5206587	5101265	3634949
GC (%)	50.61	50.35	50.75
N50	**4324771**	534858	96009
N75	4324771	359558	60713
L50	1	3	16
L75	1	6	30
# N's per 100 kbp	0.00	0.00	0.00
# predicted genes (unique)	4885	**5119**	4111
# predicted genes (>= 0 bp)	5000	5200	4111
# predicted genes (>= 300 bp)	4309	4482	2763
# predicted genes (>= 1500 bp)	664	692	317
# predicted genes (>= 3000 bp)	69	70	26

Novamente, o resultado apresentado pelo reSHAPE possui uma quantia bem menor de fragmentos do que os outros. Uma vantagem do SPAdes para essa bactéria, foi o fato de conseguir uma montagem que potencialmente possa mapear uma maior quantidade de genes. No entanto, o valor de N50 está muito aquém do tamanho total do genoma, devido a quantidade alta de fragmentos de menor tamanho.

Para a *R. sphaeroides* 2.4.1, as leituras MiSeq também podem ser encontradas no ENA: ⟨http://www.ebi.ac.uk/ena/data/view/SRR826449⟩, por sua vez as leituras PacBio foram usadas em (RIBEIRO et al., 2012), e os autores disponibilizaram esses e outros dados de genomas no site do Broad Institute: ⟨ftp://ftp.broadinstitute.org/pub/papers/assembly/Ribeiro2012/data/⟩.

Tabela 5.3: Estatísticas das montagens finais geradas para a bactéria *R. sphaeroides* 2.4.1

Assembly	reSHAPE	SPAdes	DBG2OLC
# contigs	**3**	886	129
# contigs (>= 0 bp)	3	987	129
# contigs (>= 1000 bp)	3	824	129
# contigs (>= 5000 bp)	3	309	87
# contigs (>= 10000 bp)	3	91	71
# contigs (>= 25000 bp)	3	15	49
# contigs (>= 50000 bp)	3	1	29
Largest contig	**3191234**	50571	215355
Total length	4274499	4466283	**4583420**
Total length (>= 0 bp)	4274499	4492346	4583420
Total length (>= 1000 bp)	4274499	4418941	4583420
Total length (>= 5000 bp)	4274499	3054657	4447932
Total length (>= 10000 bp)	4274499	1503527	4331298
Total length (>= 25000 bp)	4274499	495547	3950281
Total length (>= 50000 bp)	4274499	50571	3244212
GC (%)	68.85	68.51	68.47
N50	**3191234**	7529	99832
N75	949457	4184	41097
L50	1	176	15
L75	2	373	34
# N's per 100 kbp	0.00	0.00	0.00
# predicted genes (unique)	4282	**4949**	3539
# predicted genes (>= 0 bp)	4308	4950	3540
# predicted genes (>= 300 bp)	3613	3991	2545
# predicted genes (>= 1500 bp)	430	408	184
# predicted genes (>= 3000 bp)	34	29	5

Similar aos resultados da Tabela 5.2, a montagem do SPAdes se destaca em termos de número de genes estimados, ao passo que gera a montagem mais fragmentada dentre as três ferramentas analisadas. Mas nesse cenário, a montagem chega a ser quase oito vezes mais fragmentada que a do DBG2OLC, enquanto o reSHAPE produz um resultado superior às duas.

Tabela 5.4: Estatísticas das montagens finais geradas para a bactéria *F. tularensis* 99A-2628

Assembly	reSHAPE	SPAdes	DBG2OLC
# contigs	**1**	66	12
# contigs (>= 0 bp)	1	79	12
# contigs (>= 1000 bp)	1	60	12
# contigs (>= 5000 bp)	1	54	11
# contigs (>= 10000 bp)	1	46	11
# contigs (>= 25000 bp)	1	25	9
# contigs (>= 50000 bp)	1	11	6
Largest contig	**1876360**	167715	362519
Total length	**1876360**	1828406	1397070
Total length (>= 0 bp)	1876360	1832261	1397070
Total length (>= 1000 bp)	1876360	1823822	1397070
Total length (>= 5000 bp)	1876360	1805962	1395324
Total length (>= 10000 bp)	1876360	1751297	1395324
Total length (>= 25000 bp)	1876360	1361560	1356672
Total length (>= 50000 bp)	1876360	847076	1255600
GC (%)	32.22	32.22	32.88
N50	**1876360**	48424	341344
N75	1876360	24712	108481
L50	1	13	2
L75	1	26	5
# N's per 100 kbp	0.00	0.00	0.00
# predicted genes (unique)	**2136**	1928	1313
# predicted genes (>= 0 bp)	2240	2004	1313
# predicted genes (>= 300 bp)	1784	1630	903
# predicted genes (>= 1500 bp)	145	185	92
# predicted genes (>= 3000 bp)	8	16	5

Leituras MiSeq para a *F. tularensis* 99A-2628 podem ser acessadas no ENA: ⟨http://www.ebi.ac.uk/ena/data/view/SRR942045⟩. Devido ao fato desse *dataset* ser muito grande, o que pode tornar a execução do SPAdes inviável dependedo do número de *cores* disponível, um *dataset* menor na página do PBcR também pode ser utilizado: ⟨ftp://ftp.cbcb.umd.edu/pub/data/PBcR/closure_paper/FT/miseq.100X.tar.gz⟩.

Leituras PacBio também podem ser obtidas no ENA: ⟨http://www.ebi.ac.uk/ena/data/view/PRJNA212941⟩, na página do PBcR: ⟨ftp://ftp.cbcb.umd.edu/pub/data/PBcR/closure_paper/FT/filtered_subreads.200X.fastq.bz2⟩ ou no próprio repositório do reSHAPE no GitHub, que ainda disponibiliza leituras MiSeq.

Aqui tem-se o resultado mais interessante dentre todos os experimentos efetuados. Não só o reSHAPE conseguiu mais uma vez gerar uma montagem superior estatisticamente em todas as métricas (como no cenário do estudo de caso), mas essa montagem representa o genoma da *F. tularensis* 99A-2628 em apenas um contíguo.

5.2.2 Comparação com montagens do NCBI

A montagem da *F. tularensis* 99A-2628, possuindo somente um contíguo, a torna menos fragmentada e consequentemente melhor, em relação ao de números de contíguos, se comparado com a atual montagem presente no NCBI, que possui 3 contíguos. Dessa maneira, decidiu-se observar as montagens do reSHAPE em relação às respectivas montagens atuais dessas bactérias.

O reSHAPE foi capaz de diminuir o número de contíguos da *E. coli* O157:H7 str. F8092B de 9 para apenas 5, e da *R. sphaeroides* 2.4.1 de 11 para somente 3. No entanto, de forma a atestar a qualidade dessas montagens, novamente optou-se por utilizar o QUAST, dessa vez colocando lado-a-lado os resultados do reSHAPE e as montagens presentes do NCBI dessas três bactérias.

Tabela 5.5: Montagens do reSHAPE e do NCBI para a bactéria *E. coli* O157:H7 str. F8092B

Assembly	reSHAPE	NCBI
# contigs	**5**	9
# contigs (>= 0 bp)	5	9
# contigs (>= 1000 bp)	5	9
# contigs (>= 5000 bp)	5	9
# contigs (>= 10000 bp)	5	9
# contigs (>= 25000 bp)	5	8
# contigs (>= 50000 bp)	5	7
Largest contig	**4324771**	4324437
Total length	5206587	**5604406**
Total length (>= 0 bp)	5206587	5604406
Total length (>= 1000 bp)	5206587	5604406
Total length (>= 5000 bp)	5206587	5604406
Total length (>= 10000 bp)	5206587	5604406
Total length (>= 25000 bp)	5206587	5583907
Total length (>= 50000 bp)	5206587	5536411
GC (%)	50.61	50.49
N50	**4324771**	4324437
N75	4324771	4324437
L50	1	1
L75	1	1
# N's per 100 kbp	0.00	0.00
# predicted genes (unique)	4885	**5239**
# predicted genes (>= 0 bp)	5000	5363
# predicted genes (>= 300 bp)	4309	4622
# predicted genes (>= 1500 bp)	664	718
# predicted genes (>= 3000 bp)	69	80

Tabela 5.6: Montagens do reSHAPE e do NCBI para a bactéria *R. sphaeroides* 2.4.1

Assembly	reSHAPE	NCBI
# contigs	**3**	11
# contigs (>= 0 bp)	3	11
# contigs (>= 1000 bp)	3	11
# contigs (>= 5000 bp)	3	10
# contigs (>= 10000 bp)	3	9
# contigs (>= 25000 bp)	3	6
# contigs (>= 50000 bp)	3	6
Largest contig	**3191234**	3188779
Total length	4274499	**4606604**
Total length (>= 0 bp)	4274499	4606604
Total length (>= 1000 bp)	4274499	4606604
Total length (>= 5000 bp)	4274499	4604033
Total length (>= 10000 bp)	4274499	4595287
Total length (>= 25000 bp)	4274499	4545723
Total length (>= 50000 bp)	4274499	4545723
GC (%)	68.85	68.79
N50	**3191234**	3188779
N75	949457	942937
L50	1	1
L75	2	2
# N's per 100 kbp	0.00	0.00
# predicted genes (unique)	4282	**4314**
# predicted genes (>= 0 bp)	4308	4320
# predicted genes (>= 300 bp)	3613	3847
# predicted genes (>= 1500 bp)	430	540
# predicted genes (>= 3000 bp)	34	47

Nas Tabelas 5.5 e 5.6, é possivel observar que as montagens produzidas pelo reSHAPE apresentam um ganho não somente na redução do número total de contíguos, mas também no tamanho dos contíguos produzidos, onde todos são maiores que 50.000 bp. Em suma, menos contíguos são gerados com tamanho maior, ao passo que há pouca variação entre o tamanho do maior contíguo e o valor de N50 e N75 nas duas montagens.

Essa característica está relacionada com a diminuição do tamanho total das sequências, ambas na casa de centenas de milhares de pares de base, uma consequência das sobreposições (abordagem OLC) utilzada pelo GARM, que também acarreta em um menor número de genes encontrados.

O cenário da *F. tularensis* 99A-2628 (Tabela 5.7) é diferente das outras bactérias pelo fato de que foi possivel representar a sequência da mesma em apenas um único fragmento. Portanto, como observado na Tabela 5.7, a montagem gerada com o reSHAPE obteve melhores resultados em praticamente todas as estatísticas.

Tabela 5.7: Montagens do reSHAPE e do NCBI para a bactéria *F. tularensis* 99A-2628

Assembly	reSHAPE	NCBI
# contigs	**1**	3
# contigs (>= 0 bp)	1	3
# contigs (>= 1000 bp)	1	3
# contigs (>= 5000 bp)	1	3
# contigs (>= 10000 bp)	1	3
# contigs (>= 25000 bp)	1	3
# contigs (>= 50000 bp)	1	3
Largest contig	**1876360**	899925
Total length	1876360	**1877407**
Total length (>= 0 bp)	1876360	1877407
Total length (>= 1000 bp)	1876360	1877407
Total length (>= 5000 bp)	1876360	1877407
Total length (>= 10000 bp)	1876360	1877407
Total length (>= 25000 bp)	1876360	1877407
Total length (>= 50000 bp)	1876360	1877407
GC (%)	32.22	32.21
N50	**1876360**	573021
N75	1876360	573021
L50	1	2
L75	1	2
# N's per 100 kbp	0.00	0.00
# predicted genes (unique)	**2136**	1939
# predicted genes (>= 0 bp)	2240	2047
# predicted genes (>= 300 bp)	1784	1717
# predicted genes (>= 1500 bp)	145	183
# predicted genes (>= 3000 bp)	8	16

Embora a única exceção seja a diminuição no tamanho dos genes, uma quantidade maior de genes foram identificados, enquanto o tamanho total das sequências variou em apenas aproximadamente 1.000 bp, aproximadamente.

Capítulo 6

Conclusão

Avanços computacionais possibilitaram a criação de novas tecnologias de sequenciamento de nova geração, influenciando o desenvolvimento de montadores capazes de lidar com as especificidades dos dados gerados pelo processo de sequenciamento. A presente dissertação apresentou os principais problemas de sequenciamento de nova geração e suas tecnologias, estruturas de dados e abordagens utilizadas, além de uma visão aprofundada de como diversas ferramentas as implementam na prática.

Inspirado pelas ferramentas estudadas, foi desenvolvido o *pipeline* reSHAPE, um montador híbrido voltado para montagem de organismos bacterianos. Inicialmente criado para resolver o problema de montagem da *Pseudomonas aeruginosa* CCBH4851, cujo genoma é essencial para o prosseguimento de diversos estudos na Fiocruz. A partir da montagem gerada pelo reSHAPE, pesquisadores da Fiocruz e da UERJ foram capazes de finalizar o genoma da *P. aeruginosa* manualmente em apenas um contíguo, disponível no banco de nucleotídeos do NCBI com o número de acesso CP021380.

Os resultados das comparações realizadas demonstram que a ferramenta desenvolvida apresenta melhorias significativas em relação ao estado da arte de montadores híbridos quando se trata de montagem de genomas de bactérias, gerando sequências menos fragmentadas. Isso se deve ao *pipeline* do reSHAPE possibilitar o uso de montagens previamente conhecidas como sequências base para o processo de montagem, produzindo resultados mais confiáveis ao passo que melhora a montagem fornecida.

Além da produção de sequências menos fragmentadas em comparação ao SPAdes e ao DBG2OLC, também houve melhoria das montagens das três bactérias disponíveis no NCBI (em particular a *F. tularensis* 99A-2628, finalizado em apenas um contíguo), que potencialmente podem ser as melhores montagens atualmente para cada uma delas.

REFERÊNCIAS

ANTIPOV, D.; KOROBEYNIKOV, A.; MCLEAN, J. S.; PEVZNER, P. A. HybridSPAdes: an algorithm for hybrid assembly of short and long reads. *Bioinformatics*, Oxford Univ Press, p. btv688, 2015.

BAKER, M. De novo genome assembly: what every biologist should know. *Nature methods*, Nature Publishing Group, v. 9, n. 4, p. 333–337, 2012.

BANKEVICH, A.; NURK, S.; ANTIPOV, D.; GUREVICH, A. A.; DVORKIN, M.; KULIKOV, A. S.; LESIN, V. M.; NIKOLENKO, S. I.; PHAM, S.; PRJIBELSKI, A. D. et al. SPAdes: a new genome assembly algorithm and its applications to single-cell sequencing. *Journal of computational biology*, Mary Ann Liebert, Inc. 140 Huguenot Street, 3rd Floor New Rochelle, NY 10801 USA, v. 19, n. 5, p. 455–477, 2012.

BLOOM, B. H. Space/time trade-offs in hash coding with allowable errors. *Communications of the ACM*, ACM, v. 13, n. 7, p. 422–426, 1970.

BOETZER, M.; HENKEL, C. V.; JANSEN, H. J.; BUTLER, D.; PIROVANO, W. Scaffolding pre-assembled contigs using sspace. *Bioinformatics*, Oxford Univ Press, v. 27, n. 4, p. 578–579, 2011.

BOISVERT, S.; LAVIOLETTE, F.; CORBEIL, J. Ray: simultaneous assembly of reads from a mix of high-throughput sequencing technologies. *Journal of Computational Biology*, Mary Ann Liebert, Inc. 140 Huguenot Street, 3rd Floor New Rochelle, NY 10801 USA, v. 17, n. 11, p. 1519–1533, 2010.

BOLGER, A. M.; LOHSE, M.; USADEL, B. Trimmomatic: a flexible trimmer for illumina sequence data. *Bioinformatics*, Oxford Univ Press, p. btu170, 2014.

CHAPMAN, J. A.; HO, I.; SUNKARA, S.; LUO, S.; SCHROTH, G. P.; ROKHSAR, D. S. Meraculous: de novo genome assembly with short paired-end reads. *PloS one*, Public Library of Science, v. 6, n. 8, p. e23501, 2011.

CHIKHI, R.; LIMASSET, A.; JACKMAN, S.; SIMPSON, J. T.; MEDVEDEV, P. On the representation of de bruijn graphs. *Journal of Computational Biology*, Mary Ann Liebert, v. 22, n. 5, p. 336–352, 2015.

CHIN, C.-S.; ALEXANDER, D. H.; MARKS, P.; KLAMMER, A. A.; DRAKE, J.; HEINER, C.; CLUM, A.; COPELAND, A.; HUDDLESTON, J.; EICHLER, E. E. et al. Nonhybrid, finished microbial genome assemblies from long-read smrt sequencing data. *Nature methods*, Nature Research, v. 10, n. 6, p. 563–569, 2013.

COCK, P. J.; FIELDS, C. J.; GOTO, N.; HEUER, M. L.; RICE, P. M. The sanger FASTQ file format for sequences with quality scores, and the Solexa/Illumina FASTQ variants. *Nucleic acids research*, Oxford University Press, v. 38, n. 6, p. 1767–1771, 2009.

COIL, D.; JOSPIN, G.; DARLING, A. E. A5-miseq: an updated pipeline to assemble microbial genomes from illumina miseq data. *Bioinformatics*, Oxford Univ Press, p. btu661, 2014.

COMPEAU, P. E.; PEVZNER, P. A.; TESLER, G. How to apply de Bruijn graphs to genome assembly. *Nature biotechnology*, Nature Publishing Group, v. 29, n. 11, p. 987–991, 2011.

DELCHER, A. L.; PHILLIPPY, A.; CARLTON, J.; SALZBERG, S. L. Fast algorithms for large-scale genome alignment and comparison. *Nucleic acids research*, Oxford Univ Press, v. 30, n. 11, p. 2478–2483, 2002.

DIMOND, P. F. *The Long and the Short of DNA Sequencing*. 2012. Acesso em: 24 ago. 2016. Disponível em: ⟨http://www.genengnews.com/insight-and-intelligence/the-long-and-the-short-of-dna-sequencing/77899725/⟩.

ERICKSON, B. W.; SELLERS, P. H. Recognition of patterns in genetic sequences. *Time warps, string edits, and macromolecules: the theory and practice of sequence comparison*, Addison-Wesley, 1983.

FERRAGINA, P.; GAGIE, T.; MANZINI, G. Lightweight data indexing and compression in external memory. *Algorithmica*, Springer, v. 63, n. 3, p. 707–730, 2012.

FLICEK, P.; BIRNEY, E. Sense from sequence reads: methods for alignment and assembly. *Nature methods*, Nature Publishing Group, v. 6, p. S6–S12, 2009.

GEORGANAS, E.; BULUÇ, A.; CHAPMAN, J.; OLIKER, L.; ROKHSAR, D.; YELICK, K. Parallel de Bruijn graph construction and traversal for de novo genome assembly. In: IEEE. *High Performance Computing, Networking, Storage and Analysis, SC14: International Conference for*. [S.l.], 2014. p. 437–448.

GEORGANAS, E.; BULUÇ, A.; CHAPMAN, J.; HOFMEYR, S.; ALURU, C.; EGAN, R.; OLIKER, L.; ROKHSAR, D.; YELICK, K. HipMer: an extreme-scale de novo genome assembler. In: ACM. *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*. [S.l.], 2015. p. 14.

GUREVICH, A.; SAVELIEV, V.; VYAHHI, N.; TESLER, G. Quast: quality assessment tool for genome assemblies. *Bioinformatics*, Oxford Univ Press, v. 29, n. 8, p. 1072–1075, 2013.

HALLETT, M. T. *An integrated complexity analysis of problems from computational biology*. Tese (Doutorado) — University of Victoria, 1998.

HENSON, J.; TISCHLER, G.; NING, Z. Next-generation sequencing and large genome assemblies. *Pharmacogenomics*, Future Medicine, v. 13, n. 8, p. 901–915, 2012.

JAGADISH, H.; GEHRKE, J.; LABRINIDIS, A.; PAPAKONSTANTINOU, Y.; PATEL, J. M.; RAMAKRISHNAN, R.; SHAHABI, C. Big data and its technical challenges. *Communications of the ACM*, ACM, v. 57, n. 7, p. 86–94, 2014.

KOREN, S.; WALENZ, B. P.; BERLIN, K.; MILLER, J. R.; BERGMAN, N. H.; PHILLIPPY, A. M. Canu: scalable and accurate long-read assembly via adaptive k-mer weighting and repeat separation. *bioRxiv*, Cold Spring Harbor Labs Journals, p. 071282, 2017.

LANGMEAD, B.; SALZBERG, S. L. Fast gapped-read alignment with Bowtie 2. *Nature methods*, Nature Research, v. 9, n. 4, p. 357–359, 2012.

LI, Z.; CHEN, Y.; MU, D.; YUAN, J.; SHI, Y.; ZHANG, H.; GAN, J.; LI, N.; HU, X.; LIU, B. et al. Comparison of the two major classes of assembly algorithms: overlap-layout-consensus and de Bruijn graph. *Briefings in functional genomics*, Oxford University Press, v. 11, n. 1, p. 25–37, 2012.

LIPMAN, D. J.; PEARSON, W. R. Rapid and sensitive protein similarity searches. *Science*, v. 227, n. 4693, p. 1435–1441, 1985.

MEDVEDEV, P.; GEORGIOU, K.; MYERS, G.; BRUDNO, M. Computability of models for sequence assembly. In: SPRINGER. *International Workshop on Algorithms in Bioinformatics*. [S.l.], 2007. p. 289–301.

MEDVEDEV, P.; PHAM, S.; CHAISSON, M.; TESLER, G.; PEVZNER, P. Paired de Bruijn graphs: a novel approach for incorporating mate pair information into genome assemblers. *Journal of Computational Biology*, Mary Ann Liebert, v. 18, n. 11, p. 1625–1634, 2011.

METZKER, M. L. Sequencing technologies: the next generation. *Nature Reviews Genetics*, Nature Publishing Group, v. 11, n. 1, p. 31–46, 2010.

MEYER, F. Genome sequencing vs. Moore's law: cyber challenges for the next decade. *CTWatch Quarterly*, v. 2, n. 3, 2006.

MILLER, J. R.; KOREN, S.; SUTTON, G. Assembly algorithms for next-generation sequencing data. *Genomics*, Elsevier, v. 95, n. 6, p. 315–327, 2010.

MYERS, E. W.; SUTTON, G. G.; DELCHER, A. L.; DEW, I. M.; FASULO, D. P.; FLANIGAN, M. J.; KRAVITZ, S. A.; MOBARRY, C. M.; REINERT, K. H.; REMINGTON, K. A. et al. A whole-genome assembly of drosophila. *Science*, American Association for the Advancement of Science, v. 287, n. 5461, p. 2196–2204, 2000.

NEEDLEMAN, S. B.; WUNSCH, C. D. A general method applicable to the search for similarities in the amino acid sequence of two proteins. *Journal of molecular biology*, Elsevier, v. 48, n. 3, p. 443–453, 1970.

PELL, J.; HINTZE, A.; CANINO-KONING, R.; HOWE, A.; TIEDJE, J. M.; BROWN, C. T. Scaling metagenome sequence assembly with probabilistic de bruijn graphs. *Proceedings of the National Academy of Sciences*, National Acad Sciences, v. 109, n. 33, p. 13272–13277, 2012.

PENG, Y.; LEUNG, H. C.; YIU, S.-M.; CHIN, F. Y. Idba-ud: a de novo assembler for single-cell and metagenomic sequencing data with highly uneven depth. *Bioinformatics*, Oxford Univ Press, v. 28, n. 11, p. 1420–1428, 2012.

PEVZNER, P. A.; TANG, H.; WATERMAN, M. S. An eulerian path approach to dna fragment assembly. *Proceedings of the National Academy of Sciences*, National Acad Sciences, v. 98, n. 17, p. 9748–9753, 2001.

POP, M. Genome assembly reborn: recent computational challenges. *Briefings in bioinformatics*, Oxford Univ Press, v. 10, n. 4, p. 354–366, 2009.

POP, M.; SALZBERG, S. L.; SHUMWAY, M. Genome sequence assembly: algorithms and issues. *Computer*, IEEE, v. 35, n. 7, p. 47–54, 2002.

PRJIBELSKI, A. D.; VASILINETC, I.; BANKEVICH, A.; GUREVICH, A.; KRIVOSHEEVA, T.; NURK, S.; PHAM, S.; KOROBEYNIKOV, A.; LAPIDUS, A.; PEVZNER, P. A. ExSPAnder: a universal repeat resolver for dna fragment assembly. *Bioinformatics*, Oxford Univ Press, v. 30, n. 12, p. i293–i301, 2014.

QUAIL, M. A.; SMITH, M.; COUPLAND, P.; OTTO, T. D.; HARRIS, S. R.; CONNOR, T. R.; BERTONI, A.; SWERDLOW, H. P.; GU, Y. A tale of three next generation sequencing platforms: comparison of ion torrent, pacific biosciences and illumina miseq sequencers. *BMC genomics*, BioMed Central, v. 13, n. 1, p. 1, 2012.

RIBEIRO, F. J.; PRZYBYLSKI, D.; YIN, S.; SHARPE, T.; GNERRE, S.; ABOUELLEIL, A.; BERLIN, A. M.; MONTMAYEUR, A.; SHEA, T. P.; WALKER, B. J. et al. Finished bacterial genomes from shotgun sequence data. *Genome research*, Cold Spring Harbor Lab, v. 22, n. 11, p. 2270–2277, 2012.

SAMANTA, M. *From Multiple Kmers to Multi-kmer de Bruijn Graph*. 2012. Acesso em: 20 jun. 2015. Disponível em: ⟨http://www.homolog.us/blogs/blog/2012/10/10/multi-kmer-de-bruijn-graphs/⟩.

SCHBATH, S.; MARTIN, V.; ZYTNICKI, M.; FAYOLLE, J.; LOUX, V.; GIBRAT, J.-F. Mapping reads on a genomic sequence: an algorithmic overview and a practical comparative analysis. *Journal of Computational Biology*, Mary Ann Liebert, v. 19, n. 6, p. 796–813, 2012.

SILVEIRA, M.; ALBANO, R.; ASENSI, M.; ASSEF, A. P. C. The draft genome sequence of multidrug-resistant pseudomonas aeruginosa strain CCBH4851, a nosocomial isolate belonging to clone SP (ST277) that is prevalent in Brazil. *Memórias do Instituto Oswaldo Cruz*, SciELO Brasil, v. 109, n. 8, p. 1086–1087, 2014.

SIMPSON, J. T.; DURBIN, R. Efficient de novo assembly of large genomes using compressed data structures. *Genome research*, Cold Spring Harbor Lab, v. 22, n. 3, p. 549–556, 2012.

SIMPSON, J. T.; WONG, K.; JACKMAN, S. D.; SCHEIN, J. E.; JONES, S. J.; BIROL, I. ABySS: a parallel assembler for short read sequence data. *Genome research*, Cold Spring Harbor Lab, v. 19, n. 6, p. 1117–1123, 2009.

SMITH, T. F.; WATERMAN, M. S. Identification of common molecular subsequences. *Journal of molecular biology*, Elsevier, v. 147, n. 1, p. 195–197, 1981.

SOTO-JIMENEZ, L. M.; ESTRADA, K.; SANCHEZ-FLORES, A. GARM: genome assembly, reconciliation and merging pipeline. *Current topics in medicinal chemistry*, Bentham Science Publishers, v. 14, n. 3, p. 418–424, 2014.

STEIN, L. D. The case for cloud computing in genome informatics. *Genome biology*, BioMed Central, v. 11, n. 5, p. 1, 2010.

TRITT, A.; EISEN, J. A.; FACCIOTTI, M. T.; DARLING, A. E. An integrated pipeline for de novo assembly of microbial genomes. *PloS one*, Public Library of Science, v. 7, n. 9, p. e42304, 2012.

UKKONEN, E. Algorithms for approximate string matching. *Information and control*, Elsevier, v. 64, n. 1-3, p. 100–118, 1985.

UTTURKAR, S. M.; KLINGEMAN, D. M.; LAND, M. L.; SCHADT, C. W.; DOKTYCZ, M. J.; PELLETIER, D. A.; BROWN, S. D. Evaluation and validation of de novo and hybrid assembly techniques to derive high-quality genome sequences. *Bioinformatics*, Oxford Univ Press, v. 30, n. 19, p. 2709–2716, 2014.

VALLET-GELY, I.; BOCCARD, F. Chromosomal organization and segregation in pseudomonas aeruginosa. *PLoS Genet*, Public Library of Science, v. 9, n. 5, p. e1003492, 2013.

VICEDOMINI, R.; VEZZI, F.; SCALABRIN, S.; ARVESTAD, L.; POLICRITI, A. GAM-NGS: genomic assemblies merger for next generation sequencing. *BMC bioinformatics*, BioMed Central, v. 14, n. 7, p. S6, 2013.

WENCES, A. H.; SCHATZ, M. C. Metassembler: merging and optimizing de novo genome assemblies. *Genome biology*, BioMed Central, v. 16, n. 1, p. 207, 2015.

YE, C.; HILL, C. M.; WU, S.; RUAN, J.; MA, Z. S. DBG2OLC: efficient assembly of large genomes using long erroneous reads of the third generation sequencing technologies. *Scientific Reports*, Nature Publishing Group, v. 6, 2016.

ZIMIN, A. V.; SMITH, D. R.; SUTTON, G.; YORKE, J. A. Assembly reconciliation. *Bioinformatics*, Oxford Univ Press, v. 24, n. 1, p. 42–45, 2008.