Metodologias para Construção de Genomas

André Luiz de Oliveira

Tópicos em Bioinformática

Prof. Dr. Allan Mitchell Durham

Instituto de Matemática e Estatística

Nota do autor: meu interesse nos montadores desenhados para trabalhar com os dados originados pelos sequenciadores de nova geração ou massivamente paralelos, me acompanha desde o inicio de minha Pós-Graduação. Espero que no processo de construção dessa resenha alguns conceitos se tornem mais claros para mim além de entender de uma maneira mais aprofundada as dificuldades nos processos de montagem de genomas. Boa leitura!

Introdução:

- Sequenciamento e montagem: Frederick Sanger (que recebeu o prêmio Nobel de Química em 1958 e 1980) mudou a cara da biologia moderna fornecendo ferramentas que ajudaram a caracterizar os sistemas biológicos. O sequenciamento Sanger (Sanger 1977) se baseia em cadeias de terminação específicas apoiadas em quatro reações separadas ("A", "T", "C" e "G"). As inovações propostas por Sanger são: uso de didesoxinucleosídeos, uso de isótopos radioativos detectáveis em radiografia e o uso da policrilamida para separar e detectar individualmente cada molécula de DNA sequenciado.

Ao longo de 30 anos essa metodologia foi e ainda é utilizada por grandes Centros de Genômicas ao redor do mundo e é considerada como o *"gold standard"* para realização do sequenciamento. Porém existem algumas limitações relacionadas a essa metodologia (Schadt, Kasarkis 2010):

1. Como a metodologia é baseada na clonagem de um DNA exógeno em um vetor, o vetor precisa ser capaz de incorporar esse DNA e que esse material incorporado seja compatível com a máquina de replicação desse vetor (ex: BACs);
2. Algumas partes do cromossomo sequenciado são praticamente inclonáveis dificultando caracterizar essas regiões.
3. Por ultimo podemos dizer que o custo de uma base sequenciada por essa metodologia é absurdamente caro.

Juntamente com o sequenciamento surgiram outros problemas para serem supridos como, por exemplo, a montagem dos genomas. Todas as tecnologias de sequenciamento partilhar de uma mesma limitação fundamental: os tamanhos dos *reads* gerados pela técnica são muito menores do que o tamanho do menor genoma existente é necessário montá-los. Assim ferramentas computacionais têm sido desenvolvidas para eficientemente montar os fragmentos de DNA, como irei descrever algumas ao longo da resenha.

Para essa montagem ser realizada é necessário que as bases presentes nos *reads* apresentem uma boa qualidade, os programas necessitam ter capacidade para lidar com um grande volume de dados e os algoritmos tem que ser maleáveis o suficiente para evitar montagens erradas caso o genoma sequenciado apresente muitas regiões repetidas ou duplicadas. A montagem também depende diretamente do método de sequenciamento, que por sua vez depende do organismo que está sendo sequenciado (Scheibye-Alsing, Gorodkin 2010).

- Um pouco sobre o sequenciamento de nova geração: Quando pensávamos que a era do sequenciamento havia passado o seu pico as novas tecnologias de sequenciamento trouxeram um novo potencial e uma gama muito maior de aplicações que já estão transformando as pesquisas biológicas. **(**Marguerat, Bähler 2008). Dois trabalhos pioneiros deram o primeiro vislumbre do que estaria por vir (Shendure, 2005 e Margulies, 2005) Essa revolução é dirigida

por três plataformas hoje em dia: 454 (Roche), Genome Analyzer (Illumina/Solexa) e a ABI-SOLiD (Apllied Biosystem).

Essas metodologias por se basearem em uma metodologia Sanger citada anteriormente, são comumente chamadas de "nova geração", ou sequenciadores massivamente paralelos. Todas essas tecnologias são distintas, porém tem algo em comum, geram quantidades extremas de dados de saída em uma escala nunca vista antes, sem necessitar de bibliotecas de clones, e o principal atrativo: o custo por base sequenciada é muito menor que a metodologia Sanger. (Janitz, 2008).

Porém com essas novas tecnologias novos problemas foram gerados e juntamente com o preço da base sequenciada o tamanho do *read* diminuiu consideravelmente, de ~ 1000 pb gerado pela metodologia Sanger para o intervalo de ~400 pb (454 Titanium) até ~25-50 pb (SOLiD). Abaixo está uma tabela destacando os impactos da diminuição desses *reads* no processo de montagem (Pop, 2009):

Tabela 1: Características dos dados originados pelos novos sequenciadores e o seu impacto na montagem de genomas:

Características do NGS	**Desafio para Montagem**
Reads pequenos	Dificuldade em montar repetições
Novos tipos de erros	Necessidade de modificar os softwares para incorporar essas novas características
Grande quantidade de Dados	Necessita de implementações em paralelo e hardware especializado

Antes de introduzir as duas metodologias mais usadas para a montagem de genomas descreverei sucintamente de um procedimento que é realizado durante o processo de sequenciamento que tem ajudado bastante na montagem de genomas.

- Reads paired-end*:* são usados frequentemente para validar experimentalmente o processo de montagem (pode ser chamado também de *reads mate pairs*). Durante o processo de sequenciamento esses *read pairs* são produzidos sequenciando ambas as extremidades do fragmento de DNA. O aumenta da quantidade de dados de saída assim como a cobertura dos *reads* facilita a montagem *de novo* assim como a identificação de *indels,* inversões e outros arranjos genômicos. (Janitz, 2008). O uso de *paired-ends,* como irá ser mostrado ao longo da resenha, fornece uma boa arquitetura para a resolução de ambiguidades ao longo do processo de *scaffolding* nesses novos montadores.

- Overlap-Layout-Consensus vs Grafos de Bruijn: Se apropriando de uma metáfora simples introduzida por Pevzner (2001): "Crianças gostam de quebra-cabeças, e elas geralmente os

montam pela tentativa de combinação de cada par de pedaços os unindo quando a correspondência". Os biólogos tentam montar genomas de uma forma muito similar, com uma única diferença que o número de peças do quebra-cabeça dos biólogos é muito maior que o da criança. Essa abordagem de comparação entre todas as possíveis peças se tornou inviável quando resolveram aumentar ainda mais o número de peças do quebra cabeça (surgimento dos novos sequenciadores) e tornou-se necessário o desenvolvimento de programas que conseguissem manejar essa quantidade excessiva de dados. Surgiu então um novo paradigma que corresponde aos montadores que realizam uma montagem *todos-contra-todos* e os demais que tentam aperfeiçoar essa montagem.

- ***Overlap-layout-consensus:*** Dentre os algoritmos que seguem essa abordagem os mais conhecidos são: o CAP3 (Huang, Madan 1999) com mais de 445 citações no Pubmed e o Phrap (Gordan, Green 1998). Esses algoritmos seguem alguns passos específicos:

i. A qualidade das sequências é inspecionada e há uma remoção dos *reads* errados;
ii. Existe a computação das sobreposições entre todas as sequências;
iii. As falsas sobreposições são retiradas;
iv. Os *contigs* são construídos;
v. É realizado um múltiplo alinhamento de sequências e geração da sequencia consenso.

- *Fase do overlap:* uma identificação das sobreposições é executada pelo algoritmo para identificar subsequências exatas (***K-mers***) entre os *reads*. Essas subsequências idênticas são usadas para encontrar pares de sobreposições entre as sequencias, nos quais são alinhadas umas as outras de modo a checar se essa sobreposição é realmente verdadeira. Para a maioria desses algoritmos é utilizado para realizar o alinhamento é uma modificação do algoritmo original Smith-Waterman (Smith, Waterman 1981).

A ideia básica desse algoritmo é utilizar de programação dinâmica para construir uma matriz contendo os escores de todas as subsequências, que é analisada e encontra o alinhamento ótimo (Scheibye-Alsing, Gorodkin 2010). Essa informação é usada para construir um grafo de sobreposições que contém uma aresta conectando dois nós se alguma sobreposição foi identificada entre cada *read*.

- *Fase do layout:* durante essa etapa, o grafo de sobreposições é simplificado pela eliminação de redundância entre os *reads* e o grafo é analisado para identificar caminhos que correspondem aos segmentos do genoma que está sendo montado. Montar esses segmentos do genoma corresponde achar o caminho onde o grafo visita cada vértice apenas uma vez, isso é conhecido como circuito Hamiltoniano. Encontrar esse circuito é um problema NP-completo, o que significa que é efetivamente impossível calcular uma ótima solução (Paszkiewicz 2010).

Os programas como CAP3 e PHRAP utilizam heurísticas para lidar com esse problema, e essas heurísticas são conhecidas como "estratégias egoístas" para gerar o layout a partir do conjunto de sobreposições. Essa estratégia consiste em progressivamente fundir pares de *strings* que apresentam os maiores escores de sobreposição até que apenas uma única *string*

reste. Essa aproximação apesar de não ser necessariamente a melhor solução global para o problema provê resultados bons.

-Fase do consensus: Por último o algoritmo constrói um alinhamento múltiplo de sequências a partir da qual a sequência consenso é obtida. Todos os *reads* encontrados ao longo do caminho Hamiltoniano são recrutados e ocorre um alinhamento global que dá origem a sequência consenso.

É possível ver que na fase do layout assim como a fase consenso a demanda computacional é intensa tornando inviável tentar montar genomas originados pelas novas tecnologias de sequenciamento. A falta de um algoritmo eficiente para a solução dessas duas fases pode ser contornada utilizando uma abordagem baseada em grafos de Bruijn ou em caminhos Eulerianos.

-Grafos de Bruijn: A abordagem Euleriana foi descrita pela primeira vez por Pevzner (Pevzner 2001). Os algoritmos baseados nessa abordagem contrastam com o descrito acima por não tentarem encontrar jamas qualquer correspondência entre os pares de sequencias, abdicando da fase do layout, ao invés disso eles cortam as sequencias em pedaços ainda menores de tamanho regular (***K-mer***) e essa operação transporta a montagem do genoma do mundo complicado Hamiltoniano para dentro do mundo Euleriano, com algoritmos polinomiais para realizar essa montagem.

Para explicar a abordagem Euleriana nada melhor do que explicar como os algoritmos que a utilizam funcionam e para isso selecionei quatro dos mais recentemente programas que implementam essa abordagem e discorrerei sobre as diferenças e as semelhanças entre as heurísticas de cada um deles. Um aspecto importante que levei em consideração ao escolher os algoritmos foi o número de citações dos mesmos fornecidos pelo Pubmed (Tabela 2).

Tabela 2: Nome do montador para dados de nova geração e as suas citações no Pubmed

Montador	Citações no Pubmed
Velvet	103
ABySS	16
EULER-SR	35
ALLPaths	34
SOAPdenovo	10

Recentemente tive a oportunidade de realizar um minicurso com o desenvolvedor do AllPaths (Buttler, Jaffe 2008 e MacCallum, Jaffe 2009) e ficou claro que a metodologia utilizada por esse montador é diferente em essência dos grafos de Bruijn (citada como *unipaths* no trabalho original) além de seu escopo ser extremamente reduzido em relação a utilização dos dados do sequenciamento. Para o seu funcionamento é necessário que o sequenciamento tenha sido feito exclusivamente pela plataforma Illumina assim como seja mandatório a utilização de duas bibliotecas distintas produzidas no processo. Isso me fez optar pelo montador SOAPdenovo que recentemente teve destaque no meio acadêmico por ter sido o montador utilizado para a montagem do genoma do panda gigante (Li , Wang 2010).

O algoritmo EULER-SR (Pevzner 2001, 2004, e Chaisson e Pevzner 2008)*:* Pavel Pevzner originalmente desenvolveu o algoritmo Euler para lidar com *reads* originadas pelo sequenciamento Sanger. Em sua nova implementação EULER-SR foi desenhado para trabalhar com *reads* curtos originados pelas novas tecnologias de sequenciamento como, por exemplo, Solexa/Illumina e 454 além de ter implementações destinadas a trabalhar com bibliotecas *paired-ends* e realizar montagens *de novo* (sem utilizar nenhum genoma de referência).

O procedimento de montagem do EULER-SR é executado em vários passos: existe uma remoção dos erros derivados do sequenciamento, construção do grafo, correção do grafo, e finalmente montagem pela "transformação" dos caminhos eulerianos dentro do grafo correto em *contigs.* Seguem eles agora com maiores detalhes:

- *Correção de erros:* os erros inerentes à química e metodologia da plataforma de sequenciamento assim como a presença de inserções e deleções são inevitáveis, é imprescindível detectar esses erros nos *reads* para determinar com acurácia a sequencia finalizada. Pevzner et al. (2001) introduziu a correção de erros antes do processo de montagem e demonstrou que isso simplifica grandemente a montagem.

Seja ***s*** um read sequenciado (com erro) derivado de um genoma ***G***. Se a sequencia no genoma ***G*** for conhecida então à correção do erro no read ***s*** pode ser feita alinhamento alinhando esse read ***s*** contra o genoma ***G***. Porém na vida real, o genoma ***G*** não é conhecido até o último estágio de montagem desses dados. É um problema "catch-22" (citação): para montar um genoma é altamente desejável corrigir os erros nos *reads* primeiro, mas para corrigir os erros em um *read* é necessário montar o genoma primeiro. Para superar esse "catch-22" é utilizada uma metodologia chamada "*spectral alignment*".

Esse método leva em consideração um read ***r*** e um conjunto de ***K-mers*** $\mathfrak{S}$, chamado de espectro, e encontra o menor número de substituições, inserções, e deleções em ***r*** requeridas para tornar cada ***K-mer*** em ***r*** pertencente à $\mathfrak{S}$ ou o considera incorrigível caso muitas mudanças são requeridas. Um ***K-mer*** é considerado sólido se ele pertencer a mais do que ***M*** *reads* (onde ***M*** é o *threshold*) e fraco se estiver abaixo desse limiar. O conjunto $\mathfrak{S}$ é escolhido pela contagem das frequências de todos os ***K-mers*** presente em todos os *reads* e que apresentam multiplicidade acima do *threshold* ***m*** definido (***K-mers*** sólidos).

O "*spectral alignment"* é um processo iterativo onde o conjunto de todos *reads* e os ***K-mers*** sólidos gradualmente reduzem o número de ***K-mers*** fracos, aumentando o número de *reads* sem erros em detrimento dos *reads* com erros, levando a eliminação de muitos erros em projetos de sequenciamento. A escolha dos parâmetros apropriados para correção dos erros usando "*spectral alignment"* minimiza o número de ***K-mers*** errôneos dentro dos *reads* enquanto não afeta sequencias corretas, porém com baixa cobertura.

Essa correção de erros também é o gargalo computacional do algoritmo, EULER-NR tem um gasto de aproximadamente 12 horas para realizar essa correção e apenas ½ hora para montar o genoma todo (em uma máquina de 1.8-GHz).

É uma pratica comum entre os grandes grupos de sequenciamento descartar as sequencia com uma baixa qualidade. EULER-NR utiliza os arquivos de qualidade quando estes estão disponíveis, entretanto a rotina relacionada à correção de erros do EULER-SR consegue identificar *reads* com baixa qualidade sem os arquivos com os valores de qualidade e realizar a filtragem.

-***Construção dos grafos de Bruijn:*** Devido a grande quantidade de dados produzidos, é necessário realizar a montagem em um tempo linear ou perto do linear. Embora seja possível construir um grafo de Bruijn em um tempo linear o algoritmo que constrói os grafos é implementado em vários estágios.

Quando a construção dos grafos está sendo realizada, o tamanho dos ***k-mers*** precisam ser escolhidos para serem grandes o suficiente para evitar emaranhamento excessivo dos grafos que leva a formação de *contigs* menores. Para o EULER-SR foi encontrado que o valor de ***k*** = 35 remove a maioria dos emaranhados com um custo de uma pequena fragmentação na montagem.

- *Correção dos grafos de Bruijn:* Entretanto mesmo após a remoção dos erros derivados do sequenciamento, um pequeno número de erros representados por arestas adicionais na topologia do grafo ainda precisam ser corrigidos. EULER-NR através de cálculos e aproximações derivados dos *reads* gera um grafo hipotético, chamado de grafo G, que representa perfeitamente o genoma construído. Por exemplo, se houver uma pequena mudança de base decorrente do processo de sequenciamento isso irá criar uma aresta (***K + 1***) adicional dentro do grafo. Portanto o algoritmo EULER-NR irá executar a correção destes erros adicionais pela detecção e remoção destas arestas erradas realizando a comparação com o Grafo G. A principal meta dessas correções é simplificar o grafo até que todas as arestas errôneas sejam removidas e nada mais. Isso é diferente da simplificação do grafo feito pelo EULER-SR no final do processo de montagem.

Além disso, outras técnicas e heurísticas adicionais são usadas para remover erros comuns relatados aos novos sequenciadores como: (1) erros comuns no final dos *reads* na posição 3'; (2) erros ocasionados por sequencias em *tandem*; (3) erros potencias devido a cobertura do sequenciamento e (4) formação de *reads* quiméricos.

Quando todos os procedimentos para reduzir os erros foram aplicados, o EULER-SR realiza uma transformação para resolver os *repeats* no grafo que são menores do que o tamanho do read. Essas repetições quando apresentam alta similaridade são fundidas dentro de uma única aresta correspondendo a sequencia consenso repetida. Um grafo de Bruijn correto contém um caminho único separado para cada sequencia repetida.

O algoritmo Velvet (*Zerbino, Birney 2008 e Zerbino, Birney 2009)* : De todos os montadores é o mais flexível podendo ser utilizado para montagem *de novo* de dados originados por quase todas as plataformas de sequenciamento, inclusive com o "colour-space" originado pela plataforma SOLiD (é o montador que mais darei atenção no trabalho). Realiza uma montagem *de novo* e é especialmente desenhado para *short-reads*. Foi desenvolvido por Daniel Zerbino e

Ewan Birney no EMBL-EBI. As etapas de montagem dos grafos de Bruijn implementado pelo Velvet consistem em dois passos: (1) montagem e construção dos grafos seguindo (2) por uma posterior correção de erros, contrastando com o algoritmo EULER-SR que realiza a correção antes da montagem.

- Estrutura e Construção dos grafos de Bruijn:

Estrutura: No grafo cada nó ***N*** representa uma série de ***K-mers*** sobrepostos. ***K-mers*** adjacentes se sobrepõem por ***k – 1*** nucleotídeos. Velvet obtém como informação de cada ***K-mer*** apenas o último nucleotídeo, chamado de ***s(N)***. Para cada nó ***N*** o algoritmo anexa um nó gêmeo ***Ñ*** (e a união de ***N*** com ***Ñ*** é chamado de bloco) que representa o reverso complementar desse ***K-mer***, garantindo que as sobreposições entre os *reads* da fita oposta sejam levados em consideração na hora da montagem (ver figura 1). Os nós podem ser conectados diretamente por um arco e devido a simetria desses blocos, Velvet utiliza uma regra simples baseado na simetria desses blocos: Se um arco sai do nó ***A***, e segue para ***O***, então um arco simétrico de ***Ã*** segue em direção para ***Õ***.

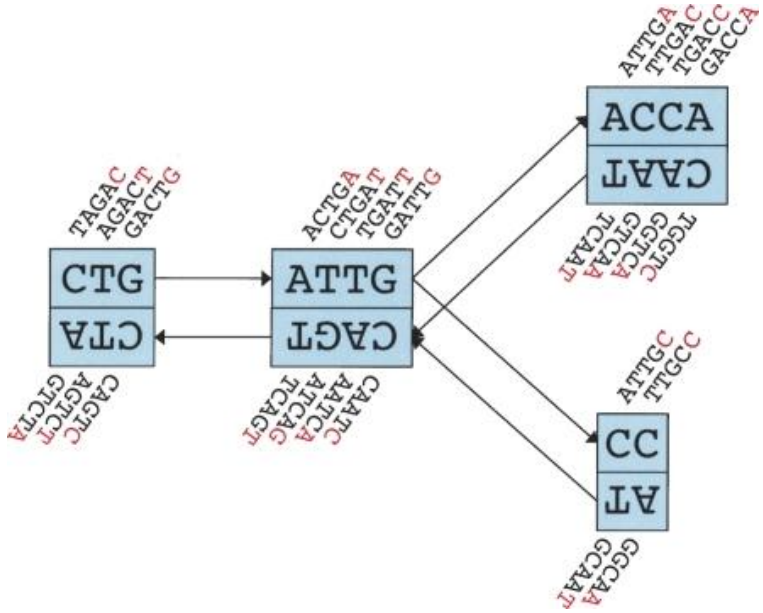

Figura 1: *Representação esquemática da implementação dos grafos de Bruijn dentro do escopo do Velvet. Cada nó é representado por um simples retângulo, no qual representa uma série de k-mers sobrepostos (nesse caso, k = 5), listado abaixo do retângulo ou ao lado. A sequência final dos nucleotídeos, copiada com letras grandes no retângulo, é a sequência representativo do nó. O nó gêmeo, diretamente ligado ao nó, representa a série de sequências reversas e complementares dos k-mers. Os Arcos são representados como flechas entre os nós. O último k-mer da origem do arco se sobrepõe com o primeiro de seu destino. Cada arco apresenta um arco simétrico.*

Construção: Os reads são armazenados em um *hash* de acordo com o tamanho do ***K-mer*** pré-definido pelo usuário, Velvet sugere para micro *reads* com 25 pb de tamanho um valor de ***K = 21. K-mers*** pequenos aumentam a conectividade do grafo e consequentemente a possibilidade de haver sobreposições entre dois *reads* assim, porém aumenta a possibilidade de haver

repetições ambíguas. Não só o Velvet, mas como todos os montadores baseados nessa metodologia e referidos nesse ensaio exibem um balanço delicado entre sensibilidade e especificidade dado pelo valor de ***K***. Para cada ***K-mer*** observado dentro do conjunto de *reads*, um *hash table* armazena o *ID* do primeiro *read* encontrado contendo o ***k-mer*** e a posição desse ***k-mer*** dentro do *read.* Uma maneira elegante e inteligente que o Velvet tem em evitar que os ***K-mers*** sejam os seus próprios reverso complementar e prevenir construções errôneas é permitir que o valor de ***K*** seja restritamente ímpar (se o usuário utilizar um valor par, Velvet irá decrescer automaticamente o valor e executar o programa normalmente). Essa nova representação das sequências nos reads é chamada de "roadmap".

Simplificação: após a construção do grafo, existe uma simplificação do mesmo sem nenhuma perda de informação. Os blocos são interrompidos toda a vez que um *read* termina ou inicia levando a formação de cadeias de blocos ou sub-grafos conectados e é através desses sub-grafos que ocorre a simplificação. A simplificação é realizada da seguinte forma: toda a vez que um nó ***A*** tem apenas uma arco e esse arco se direciona a um nó ***B*** que contém por sua vez apenas um único arco direcionado para ele, os dois nós são fundidos e iterativamente as cadeias de blocos são colapsadas dentro de blocos únicos.

- Remoção de erros: Ao contrário do EULER-NR a remoção dos erros é realizada durante a montagem do grafo. No Velvet os erros podem ocorrem devido ao processo de sequenciamento ou devido à amostra biológica, por exemplo, polimorfismos. Para distinguir erros de sequenciamento de polimorfismos verdadeiros, Velvet realiza um procedimento que verifica a cobertura esperada de sequencias verdadeiras em contraste com aquelas sequências que apresentam erros aleatórios. A abordagem que o próprio Velvet chama de ingênua é útil porque é esperado que erros aleatórios nos *reads* sequenciados não apresentem uma cobertura tão grande quanto à esperada para *reads* sem erros.

Porém a verdadeira remoção dos erros ocorre através da análise da topologia do grafo. Os dados construídos podem originar três tipos de "anomalias" estruturais:

i. ***Pontas (tips):*** representa uma cadeia de nós que é desconectada em um final. Para a remoção dessas pontas são considerados dois parâmetros: o tamanho e menor contagem. Uma ponta vai ser apenas removida do grafo (implicando apenas em mudanças locais e nenhuma conectividade é interrompida ao longo da estrutura) se ela for menor do um limite de corte arbitrário de ***2K***. O outro parâmetro, contagem mínima, é uma propriedade onde localizada no nó onde as pontas se conectam com o resto do grafo. É assumido que o arco que leva até a ponta tem uma multiplicidade inferior do que ao menos um dos outros arcos que estão irradiando para a junção do mesmo nó, ou seja, começando a partir de um nó, e caminhando de uma ponta é uma alternativa a um caminho mais comum. A iteração vai acontecendo até não houver mais pontas no grafo e então o grafo é simplificado novamente.

ii. ***Bolhas (bubbles)***: essas bolhas são removidas utilizando o módulo chamado "*Tour Bus*". Dois caminhos são considerados redundantes se eles começam e terminam no mesmo nó e contém sequencias similares. O *Tour Bus* (ver figura 2) trabalha sobre dois

critérios, identidade de sequências e tamanho do *thresholds*. Esse processo tenta encontrar sempre o menor caminho entre dois nós podendo fundir dois caminhos caso sejam redundantes. Isso implica em impactos complexos em termos de processamento e estrutura computacional porque requer um mapeamento total do grafo enquanto ele é simplificado e remontado.

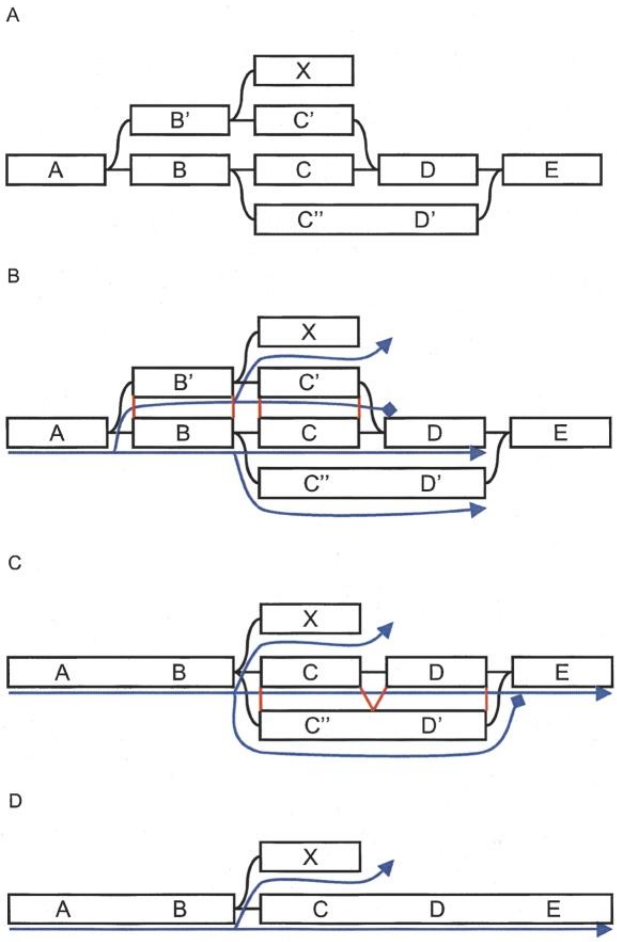

Figura 2: *Exemplo de como o algoritmo Tour Bus corrige os erros. (A) é o grafo original. (B) A busca inicia-se a partir do A e se espalha para a direita. A progressão do caminho mais superior é interrompida porque D foi previamente visitado. A sequência de nucleotídeos correspondendo ao caminho alternativo B´C´ e BC são extraídas do grafo, alinhados e comparados. (C) Os dois caminhos são julgados similares, então o maior, B´C´, é fundido dentro do caminho menor, BC. A fusão é dirigida pelo alinhamento da sequencia consenso, indicado pelas linhas vermelhas em B. Note que o nó X, no qual estava conectado em B´, é agora conectado diretamente em B. A busca continua, e a parte inferior do caminho (através de C´D´) e CD são comparados. (D) CD e C´D´ são julgados similares. E o caminho maior é fundido dentro do menor. Ocasionando assim a simplificação do grafo sem nenhuma perda de informação.*

iii. ***Removendo conexões erradas:*** Isso é realizado após o módulo *Tour-Bus* ser executado. Esses erros de conexão geram estruturas irreconhecíveis e podem ser

facilmente identificadas a partir da topologia do grafo. Então o Velvet as remove com um ponto de corte associado à cobertura da amostra de sequenciamento.

As simplificações ocorrem ao longo da execução do programa e existe um módulo adicional para lidar com as estruturas repetidas intrínsecas no genoma sequenciado além de lidar com *reads* longos presentes em sua amostra, no caso de você incorporar bibliotecas mistas. As repetições são um problema porque, para estender corretamente os *contigs* e criar *scaffolds* é necessário resolver essas repetições de um modo eficaz e isso é realizado pelo módulo "*Breadcrumb*". Através da identificação de adicionais conexões erradas, e a detecção de regiões de baixa complexidade o módulo *Breadcrumb* destrói todos os erros produzindo o grafo final e consequentemente o melhor caminho euleriano (*contig* ou *scaffold* final).

Em uma publicação mais recente Velvet, inclui mais duas novas heurísticas dentro do escopo do programa, os módulos *Pebble* e R*ockBand*, que fazem uso de informação de *paired-ends reads* para resolver de maneira mais otimizada as repetições além de melhorar a formação de *scaffolds* explorando de maneira mais eficiente as informações fornecidas por *reads* longos.

- O algoritmo ABySS (*Simpson, Birol 2009)*: é um montador *de novo* especializado para montar genomas individuais grandes (**A**ssembly **By S**hort **S**equencing) utilizando sequências *paired-ends* como por exemplo de mamíferos. A inovação do ABySS em relação aos outros montadores é que ele pode distribuir a representação dos grafos de Bruijn permitindo uma computação em paralelo do algoritmo tornando-o mais eficiente. Foi desenhado por Jared T. Simpson e Inanç Birol.

O algoritmo é processado em duas etapas. Na primeira etapa, sem usar informação dos *reads paired-ends,* todos os possíveis ***K-mers*** são gerados a partir dos *reads*. Os dados obtidos dos ***K-mers*** são então processados para remover os erros dos *reads,* ramos *"dead-end"* são removidos primeiramente pelo algoritmo e os erros mais complexos representados por pequenas anomalias estruturais e bolhas são removidas em seguida resultando na construção dos *contigs* iniciais que são progressivamente estendidos até o limite. Na segunda fase a informação *paired-end* é usada para estender os *contigs* resolvendo as ambiguidades na sobreposição entre eles fundindo-os.

PRIMEIRA FASE:

-Construindo o grafo: Como o algoritmo é paralelizável os dados são carregados dentro dos grafos de Bruijn distribuídos, e nesse processo ABySS remove qualquer sequencia que contenha bases desconhecidas (representadas por "N" ou "." caso seja originado da plataforma de sequenciamento Illumina). Cada sequencia no dado de entrada ***l -mer*** é quebrado dentro de (***l – K + 1***) sobrepondo ***K-mers*** com uma janela deslizante de tamanho ***K*** ao longo da sequencia de entrada. Como uma dada sequencia é o seu reverso complementar são considerados equivalentes, a sequencia não é adicionada ao *hash table* se o reverso complementar da sequencia já estiver presente, ao contrário do Velvet que utiliza ambas as sequências para retirar informações para montar os grafos.

Assim que os ***K-mers*** forem carregados dentro dos grafos de Bruijn previamente distribuídos, a adjacência desses ***k-mers*** é computada. Para cada ***K-mers*** dentro da coleção de

sequencias uma mensagem é enviada aos oito possíveis vizinhos. Se os vizinhos existirem eles precisam apresentar uma sobreposição de ***K – 1*** com o ***K-mer*** original.

- *Removendo erros:* antes de fundir os vértices dentro de *contigs,* o grafo precisa ser simplificado ou limpos de vértices e arestas que apresentam erros de sequenciamento. A estrutura mais comum causada por erros de sequenciamento é o ramo "*dead-end"*, que é formado por *reads* que são uma mistura de ***K-mers*** corretos e incorretos.

Como as sequências incorretas são provavelmente únicas, um final de um ramo irá terminar sem extensão. Para eliminar essas estruturas, esses ramos são identificados, quando identificados o ABySS retorna até o ponto que deu origem a esse ponto ambíguo, e se esse ramo for menor do que um tamanho de *threshold* definido pelo usuário, ele é removido do grafo. A remoção desses ramos é bem sensível a escolha do tamanho do ***K-mer,*** e como foi dita anteriormente se o ***k-mer*** for muito alto, o grafo irá ser quebrado em vários pedaços dificultando assim determinar se um ramo *"dead-end"* nasce de um erro no *read* ou da falta de cobertura entre esses ***K-mers.***

Em alguns casos, erros coincidentes podem causar falsos ramos que são unidos em ambos os lados no grafo de Bruijn. Esses casos aparecem no grafo como caminhos divergentes de um mesmo local que convergem depois de ***K*** nós (nós já vimos essas estruturas antes no algoritmo Velvet, são as chamadas bolhas). Essas bolhas podem se formar devido as regiões altamente repetidas no genoma e nesses casos o ABySS irá simplificar as repetições a simples sequências. ABySS então identifica remove essas bolhas, e os *contigs* podem ser estendidos e fundidos (*Vertex merging*) sem nenhuma ambiguidade, criando os *contigs* iniciais.

SEGUNDA FASE:

-Fundindo os contigs utilizando informação paired-end: a segunda fase da montagem irá utilizar a informação disponibilizada dos *paired-end*, se estiver disponível, para resolver ambiguidades entre os *contigs.* Essa informação é usada a fim de identificar os *contigs* que podem ser conectados. Os *reads* são alinhados aos *contigs* iniciais para criar um conjunto de *contigs* com ligações, no qual é filtrado para remover ligações erradas causadas por alguns maus alinhamentos. Logo dois *contigs* são considerados ligados se ao menos ***p*** (por *default* ***p = 5***) pares são unidos aos contigs.

Como os grafos de Bruijn podem ser extremamente densos em áreas repetitivas o ABySS utiliza uma heurística para limitar o número de vértices visitados e com isso manter um limite máximo para o custo computacional ao executar essa busca. Essa busca é executada para procurar um único caminho euleriano (uma sequência de *contigs*). Esse processo é repetido para cada *contig* e o passo final une os pontos dos caminhos consistentes e gera o *contig* na montagem final.

- O algoritmo SOAPdenovo (Li, Wang 2010) : Realiza uma montagem *de novo* de grandes genomas. Em seu trabalho inicial (De novo assembly of human genomes) realizou a montagem de genomas humanos pertencentes a pessoas africanas e asiáticas. Mais recentemente foi utilizado para montagem com sucesso do genoma do panda gigante, onde foi utilizado um supercomputador com 32 núcleos e 512 Gb de memória RAM.

SOAPdenovo é especialmente desenhado para trabalhar com dados originados da plataforma Illumina reduzindo a extensibilidade de seu escopo porém não existe teoricamente uma restrição para os dados gerados pelas outras plataformas.. Outro empecilho é o processamento relativamente elevado exigido pelo algoritmo, para pequenos genomas como bactérias e fungos, é necessário que a máquina apresente no mínimo 5GB de memória. SOAPdenovo foi desenvolvido por Ruiqiang Li e Jue Ruan.

- Correção dos erros: ao contrário de todos os montadores descritos anteriormente a correção pode ser realizada antes da montagem do grafo de Bruijn ou durante o processo (existe um parâmetro para o usuário definir isso). A correção de erros antes a montagem de pequenos genomas é pouco importante desde que as conexões errôneas podem ser facilmente removidas durante o processo de montagem do grafo, porém essa correção é essencial quando se trata de genomas grandes. SOAPdenovo afirma, que para genomas grandes, essa correção diminui brutalmente o uso de memória na fase de formação do grafo, tornando possível carregar todas as sequencias disponíveis para construir o grafo.

Para os sequenciamentos com alta cobertura, o número de ***K-mers*** corretos aparece inúmeras vezes dentro do conjunto de *reads,* enquanto para sequências que contém erros os ***K-mers*** apresentam pouca frequência. O método de correção do SOAPdenovo usa a informação da frequência dos ***K-mers***. Após esse passo é construído um *hash* que armazena a frequência de todos os ***K-mers***. Caso seu genoma seja muito grande, é possível usar paralelização e dividir conjuntos de *reads* para minimizar o tempo de execução do algoritmo.

- Construção dos grafos: Para a construção dos grafos de Bruijn, cada nó representa um ***K-mer***. Dois nós que se sobrepõem por ***K – 1*** pb em *reads* presentes na vizinhança são conectados por uma aresta. Assim como todos os outros montadores baixos valores de ***K*** tornam o grafo muito complexo com milhares de arestas criadas pelas sequencias repetidas; enquanto altos valores de ***K*** podem apresentar pouca sobreposição em regiões de pouca cobertura. SOAPdenovo diz que o uso de ***k = 25*** é o melhor *tradeoff* para montagem.

- Remoção de pontas (tips), resolução de pequenas repetições e fusão de bolhas (bubbles): SOAPdenovo executa essa etapa independentemente do tamanho do genoma envolvido. A remoção de pontas (tips), ou ramos "*dead-ends"* segue a mesma lógica que o Velvet, as pontas menores do que um valor arbitrário de ***2K*** são removidas, assim como aquelas pontas que aparecem em menor frequência do que outros caminhos alternativos que são conectados a um nó comum no grafo (também seguindo a mesma lógica que o Velvet, ver contagem mínima dentro da seção "tips" ou pontas).

As pequenas sequencias repetitivas no grafo que são menores do que o tamanho do *read* podem ser resolvidas pelos caminhos dos *reads* ao longo do grafo. Para evitar más construções, SOAPdenovo apenas tenta resolver os nós repetidos que apresentam um número igual (***N)*** de arestas que chegam e saem desse nó.

Semelhante ao *"Tour Bus"* do Velvet, um algoritmo dentro do SOAPdenovo detecta as bolhas (quando ocorrem erros de sequenciamento no meio do read porém ambas as extremidades estão corretas o caminho criado pelo erro tem a forma de uma bolha dentro do grafo) e as fundem dentro de um caminho se as sequencias presentes no caminho paralelo são muito similares, ou seja há uma comparação de sequências e se houver apenas um erro de

comparação entre as bases ou menos do que quatro pares de base de diferença com um identidade de > 90 % esses caminhos são colapsados em um único.

- Construção de contigs e o processo final de montagem: Utilizando agora das informações fornecidas pelos *paired-ends reads,* SOAPdenovo começa a conectar os *contigs* originados pelo grafo. São utilizados os *paired-ends* entre dois *contigs* para pesar a ligação destes *contigs* além de estimar o tamanho da lacuna entre os dois *contigs.*

Como último passo, é usado dois procedimentos para simplificar o grafo interconectados de *contigs* e extrair caminhos lineares não ambíguos para a construção de *scaffolds.* São eles:

i. O primeiro passo é denominado linearização do sub-grafo: as linhagens transitivas compatíveis entre os grupos de *contigs* são removidas e os *contigs* restantes são fundidos dentro de um estimando com muito cuidado o tamanho das lacunas internas do mesmo.

ii. O segundo passo é a mascaramento de repetições: se um *contig* apresenta muitas ligações conectadas a ele e muitas ligações ligando ele a outros *contigs*, porém as ligações não são compatíveis, SOAPdenovo considera esse *contig* como uma estrutura repetida. E os *contigs* repetidos são mascarados ao longo do processo, juntamente com os *contigs* que eles se conectavam. Essa estratégia efetivamente gera *scaffold* (ou *super-contigs*) relativamente fácil enquanto evita entrelaçamentos.

- Tabela Comparativa: a seguir segue uma tabela comparando algumas características dos montadores acima descritos tentando identificar diferenças e semelhanças entre os mesmos. Uma pequena observação é: quase a maioria dos algoritmos descritos acima pode atuar em paralelo, porém onde a distribuição ocorre é onde reside a diferença. Se considerarmos estritamente a execução em paralelo ao nível de grafo de Bruijn apenas o ABySS realiza, porém os demais conseguem realizar a divisão de tarefas computacionalmente excessivas, como a correção de erros antes da montagem dos grafos, distribuindo em paralelo.

Algoritmo	Correção dos Erros	Suporte para quais NGS	Paralelizável	Aceita paired-ends?	Arquivos de montagem (ace, afg)
Euler-NR	Antes	454, Illumina	SIM	SIM	NÃO
Velvet	Durante	SOLiD, Illumina, 454	NÃO	SIM	SIM (AMOS)
SOAPdenovo	Customizável	Illumina	SIM	SIM	NÃO
ABySS	Antes	Illumina, 454	SIM*	SIM	NÃO

Conclusão: Vimos que apesar de todos os algoritmos utilizarem uma metodologia comum, grafos de Bruijn, as implementações e heurísticas mudam entre eles. Vimos também que as novas plataformas de sequenciamento apresentam características e erros que tentam ser mitigados por esses montadores (como erros na região final 3' linha e falhas para sequenciar região de baixa complexidade no genoma (454) , ex: poli-As).

Tanto a metodologia *overlap-layout-consensus* quando a que utiliza grafos de Bruijn são extremamente adequadas para a construção de genomas, porém ao trabalhar com uma quantidade extrema de sequencias originada pelos novos sequenciadores a comparação *todos-contra-todos* se torna inviável. Podemos citar uma exceção que é o montador Edena (Hernandez, Schrenzel 2008) que apesar de realizar montagem utilizando *reads* pequenos faz o uso da metodologia *overlap-layout-consensus.*

Uma característica importante desses montadores é que todos realizam uma montagem *de novo*, ou seja, conseguem montar os genomas sem estarem apoiados a genomas de referência. É possível agora gerar, apenas a partir de *reads* pequenos, montagens de genomas completos com pelo menos uma boa qualidade e com muito menos custo.

Baseado em minhas experiências posso dizer que em questão de flexibilidade e facilidade de execução o programa Velvet é o mais indicado. Executei algumas vezes o programa ABySS e obtive também bons resultados em relação a montagem de pequenos genomas (menor do que 11 kb). AllPaths, Euler-SR e SOAPdenovo ainda estão para serem analisados.

Referências bibliográficas:

Butler J, MacCallum I, Kleber M, Shlyakhter IA, Belmonte MK, Lander ES,Nusbaum C, Jaffe DB. ALLPATHS: de novo assembly of whole-genome shotgun microreads. Genome Res. 2008 May;18(5):810-20.

Chaisson MJ, Pevzner PA. Short read fragment assembly of bacterial genomes. Genome Res. 2008 Feb;18(2):324-30.

Gordon, D., C. Abajian, and P. Green. 1998. Consed: A Graphical Tool for Sequence Finishing. Genome Research. 8:195-202

Hernandez D., P. François, L. Farinelli, M. Osteras, and J. Schrenzel. *Genome Research.*2008. De novo bacterial genome sequencing: millions of very short reads assembled on a desktop computer.

Huang X, Madan A. CAP3: A DNA sequence assembly program. Genome Res. 1999 Sep;9(9):868-77.

Janitz, Michal (November 2008). *Next-Generation Genome Sequencing: Towards Personalized Medicine*.

Li R, Zhu H, Ruan J, Qian W, Fang X, Shi Z, Li Y, Li S, Shan G, Kristiansen K, Li S, Yang H, Wang J, Wang J. De novo assembly of human genomes with massively parallel short read sequencing. Genome Res. 2010 Feb;20(2):265-72.

Li R, Fan W, Tian G, Zhu H, He L, Cai J, Huang Q, Cai Q, Li B, Bai Y, Zhang Z,Zhang Y, Wang W, Li J, Wei F, Li H, Jian M, Li J, Zhang Z, Nielsen R, Li D, Gu W,Yang Z, Xuan Z, Ryder OA, Leung FC, Zhou Y, Cao J, Sun X, Fu Y, Fang X, Guo X,Wang B, Hou R, Shen F, Mu B, Ni P, Lin R, Qian W, Wang G, Yu C, Nie W, Wang J, Wu Z, Liang H, Min J, Wu Q, Cheng S, Ruan J, Wang M, Shi Z, Wen M, Liu B, Ren X,Zheng H, Dong D, Cook K, Shan G, Zhang H, Kosiol C, Xie X, Lu Z, Zheng H, Li Y, Steiner CC, Lam TT, Lin S, Zhang Q, Li G, Tian J, Gong T, Liu H, Zhang D, Fang L,Ye C, Zhang J, Hu W, Xu A, Ren Y, Zhang G, Bruford MW, Li Q, Ma L, Guo Y, An N,Hu Y, Zheng Y, Shi Y, Li Z, Liu Q, Chen Y, Zhao J, Qu N, Zhao S, Tian F, Wang X, Wang H, Xu L, Liu X, Vinar T, Wang Y, Lam TW, Yiu SM, Liu S, Zhang H, Li D, Huang Y, Wang X, Yang G, Jiang Z, Wang J, Qin N, Li L, Li J, Bolund L, Kristiansen K, Wong GK, Olson M, Zhang X, Li S, Yang H, Wang J, Wang J. The sequence and de novo assembly of the giant panda genome. Nature. 2010 Jan 21;463(7279):311-7.

Maccallum I, Przybylski D, Gnerre S, Burton J, Shlyakhter I, Gnirke A, Malek J, McKernan K, Ranade S, Shea TP, Williams L, Young S, Nusbaum C, Jaffe DB. ALLPATHS 2: small genomes assembled accurately and with high continuity from short paired reads. Genome Biol.2009;10(10):R103.

Margulies M, Egholm M, Altman WE, Attiya S, Bader JS, Bemben LA, Berka J, Braverman MS, Chen YJ, Chen Z, Dewell SB, Du L, Fierro JM, Gomes XV, Godwin BC,He W, Helgesen S, Ho CH, Irzyk GP, Jando SC, Alenquer ML, Jarvie TP, Jirage KB,Kim JB, Knight JR, Lanza JR, Leamon JH, Lefkowitz SM, Lei M, Li J, Lohman KL, Lu H, Makhijani VB, McDade KE, McKenna MP, Myers EW, Nickerson E, Nobile JR, Plant R, Puc BP, Ronan MT, Roth GT, Sarkis GJ, Simons JF, Simpson JW, Srinivasan M, Tartaro KR, Tomasz A, Vogt KA, Volkmer GA, Wang SH, Wang Y, Weiner MP, Yu P, Begley RF, Rothberg JM. Genome sequencing in microfabricated high-density picolitre

reactors. Nature. 2005 Sep 15;437(7057):376-80. Epub 2005 Jul 31. Erratum in: Nature. 2006 May 4;441(7089):120. Ho, Chun He [corrected to Ho, ChunHeen].

Paszkiewicz K, Studholme DJ. De novo assembly of short sequence reads. BriefBioinform. 2010 Sep;11(5):457-72. Epub 2010 Aug 19.

Pevzner PA, Tang H, Waterman MS, A New Approach to Fragment Assembly in DNA Sequencing. Proceedings of The 5th Annual International Conference on Computational Molecular Biology (RECOMB 2001), 256-267, Canada. ACM Press.

Pevzner PA, Tang H, Waterman MS, An Eulerian path approach to DNA fragment assembly. Proc. Natl. Acad. Sci. USA 2001 Aug 14; 98(17):9748-53.

Pop M. Genome assembly reborn: recent computational challenges. Brief Bioinform. 2009 Jul;10(4):354-66. Epub 2009 May 29.

Sanger F, Nicklen S, Coulson AR. DNA sequencing with chain-terminating inhibitors. 1977. Biotechnology. 1992;24:104-8.

Schadt EE, Turner S, Kasarskis A. A window into third-generation sequencing. Hum Mol Genet. 2010 Oct 15;19(R2):R227-40

Scheibye-Alsing K, Hoffmann S, Frankel A, Jensen P, Stadler PF, Mang Y, Tommerup N, Gilchrist MJ, Nygård AB, Cirera S, Jørgensen CB, Fredholm M, Gorodkin J. Sequence assembly. Comput Biol Chem. 2009 Apr;33(2):121-36.

Shendure J, Porreca GJ, Reppas NB, Lin X, McCutcheon JP, Rosenbaum AM, Wang MD, Zhang K, Mitra RD, Church GM. Accurate multiplex polony sequencing of an evolved bacterial genome. Science. 2005 Sep 9;309(5741):1728-32.

Simpson JT, Wong K, Jackman SD, Schein JE, Jones SJ, Birol I. ABySS: a parallel assembler for short read sequence data. Genome Res. 2009 Jun;19(6):1117-23.

Smith TF, Waterman MS. Identification of common molecular subsequences. J Mol Biol. 1981 Mar 25;147(1):195-7.

Velvet: algorithms for *de novo* short read assembly using de Bruijn graphs. D.R. Zerbino and E. Birney. Genome Research **18**:821-829.

Zerbino DR, McEwen GK, Margulies EH, Birney E. Pebble and rock band: heuristic resolution of repeats and scaffolding in the velvet short-read de novo assembler. PLoS One. 2009 Dec 22;4(12):e8407.