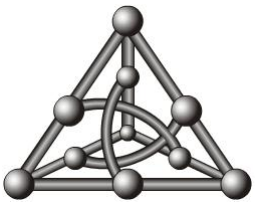
Algoritmos Genéticos e o Problema da Montagem de Reads

Dissertação de Mestrado

Joelmo Silva Fraga
(joelmo.sf@hotmail.com)



Faculdade de Computação
Universidade Federal de Mato Grosso do Sul
Caixa Postal 549 – CEP 79070-900 – Campo Grande – MS

Orientadora: Prof[a]. Luciana Montera Cheung
(montera@facom.ufms.br)

Campo Grande, Novembro de 2014

Algoritmos Genéticos e o Problema da Montagem de Reads

Este exemplar corresponde à redação final da dissertação devidamente corrigida e defendida por Joelmo Silva Fraga e aprovada pela Banca Examinadora.

Campo Grande, 05 de Novembro de 2014.

Banca Examinadora:

- Profª. Drª. Luciana Montera Cheung (FACOM/UFMS) - orientadora
- Prof. Dr. Francisco Eloi Soares de Araujo (FACOM/UFMS)
- Prof. Dr. Said Sadique Adi (FACOM/UFMS)

Agradecimentos

Agradeço a Deus pelo dom da vida e por nos proporcionar salvação a todos nós, à minha orientadora professora Luciana Montera Cheung por todos os momentos de auxílio e suporte destinado a este projeto, sem a qual não seria possível ser finalizado. Agradeço a todos os meus amigos e familiares que contribuíram de forma direta ou indireta no decorrer deste projeto. Agradeço à minha esposa Izabela por toda a compreensão e companheirismo dedicado a mim durante este período, à minha querida mãe, Maria Celma por me apoiar e incentivar durante toda a minha vida e não me deixar desistir. Ao CNPQ por todo auxílio financeiro.

Resumo

O problema de montagem de *reads* é um problema da Bioinformática considerado de grande complexidade devido à sua característica combinatória e ao fato de ser um problema dependente das tecnologias de sequenciamento. *Reads* são fragmentos de DNA e o processo de montagem consiste, idealmente, na obtenção de uma única sequência de DNA a partir deste conjunto de fragmentos. São encontradas na literatura diferentes abordagens para a realização da montagem. Dentre elas destacam-se aquelas baseadas em grafos de sobreposição e de Bruijn e as estratégias gulosas. Heurísticas estão sendo exploradas, tais como *Simulated annealing* (arrefecimento simulado), *Scartter search* (busca tabu) e Algoritmo Genético (GA). Este trabalho apresenta um modelo e uma implementação para o problema de montagem de *reads* através de um algoritmo genético. Os resultados mostram que este modelo é capaz de realizar a montagem e demonstram como o modelo se comporta mediantes os parâmetros estabelecido para sua execução.

Abstract

The DNA Fragment Assembly Problem is a Bioinformatics problem considered as a problem of large complexity due to its combinatorial characteristic and to the fact it being a problem that depends on the sequencing technologies. Reads are DNA fragments and the assembly process consists, ideally, in obtaining a DNA sequence from this set of fragments. There are different approaches in the literature to make the assembly. Among them are those based on overlay graphs, Bruijn graphs and greedy strategies. Heuristics are being explored, such as Simulated Annealing, Scatter Search and genetic algorithms. This work presents a model and an implementation for the DNA Fragment Assembly Problem using a genetic algorithm. The results show that this model is capable of performing the assembly and demonstrates how the model behaves using parameters established for its running.

Sumário

Lista de Figuras 8

Lista de Tabelas 10

1 Introdução 11

1.1 Sanger e NGS . . . 11
1.2 Justificativa . . . 15
1.3 Objetivo . . . 15
1.4 Organização do Texto . . . 15

2 Problema de Montagem 16

2.1 Definição do Problema de Montagem de Reads . . . 18
2.2 Considerações . . . 19
2.3 Estratégias de Montagem . . . 20
2.3.1 Grafo de Bruijn . . . 20
2.3.2 Grafo de Sobreposição . . . 21
2.3.3 Algoritmos Gulosos Baseado em Grafos . . . 23
2.4 N50 . . . 24

3 Algoritmo Genético 26

3.1 História . . . 26
3.2 Conceitos e Funcionamento de um AG . . . 26
3.3 Trabalhos Relacionados . . . 31

4 Algoritmos Genéticos e a Montagem de Reads 33

4.1 Representação de um Indivíduo . . . 33

4.2 Avaliação 34
4.3 Operadores Genéticos 36
4.4 Algoritmo 38

5 Implementação, Testes e Resultados **40**
5.1 Sobre a Implementação do Modelo 40
5.2 Sobre as Ferramentas 41
5.2.1 ART 41
5.3 Testes e Resultados 41

6 Conclusões e Trabalhos Futuros **48**

Referências Bibliográficas **49**

Lista de Figuras

1.1 Esquema do processo de sequenciamento e montagem de *reads* adaptado de [22]. 12
1.2 *Pipeline* do processo de sequenciamento das NGS, adaptado de [35]. 13

2.1 Ilustração de um trecho de molécula de DNA. 16
2.2 Sobreposição entre os *reads* r_1 e r_2. 17
2.3 Alinhamentos entre os *reads* r_5 e r_6. 17
2.4 Sequência consenso resultante do alinhamento entre os *reads* r_1 a r_6. 18
2.5 Ocorrência de regiões repetidas. 19
2.6 Exemplo de cobertura incompleta. 20
2.7 Grafo de Bruijn para o conjunto R. 21
2.8 Exemplo de um grafo de sobreposição com 5 *reads*. 22
2.9 layout para o grafo de sobreposição construído na etapa anterior. 22
2.10 Sequência consenso que representa o DNA alvo montado a partir dos *reads*. . 23
2.11 Montagem baseada em algoritmo guloso para o conjunto de *reads* R. 23
2.12 Calculo do N50 para um conjunto de *contigs*. 25

3.1 Esquema de funcionamento de um Algoritmo Genético. 27
3.2 Crossover de um ponto e Mutação. 28
3.3 Crossover de dois pontos. 29
3.4 Exemplo da seleção por roleta para a Tabela 3.2. 30

4.1 Representação de um indivíduo composto por $k = 16$ genes. 33
4.2 Representação e interpretação do indivíduo no modelo proposto. 34
4.3 Processo de avaliação de um indivíduo. 35
4.4 Exemplo de *reads* contidos 35

4.5 Operador de *crossover* sendo aplicado em um par de indivíduos. 36
4.6 Operador de mutação sendo aplicado em um indivíduo 36
4.7 Exemplo da mutação promovendo um novo indivíduo. 37

5.1 Diagrama de fluxo do modelo. 40
5.2 *Layout* do conjunto de teste 1. 41
5.3 Influência no número de iterações internas e externas sobre a montagem de *reads*. 43
5.4 Disposição dos reads do segundo e terceiro caso de teste. 43
5.5 Influência no número de iterações internas e externas sobre a montagem do caso de teste 2 de *reads*. 46
5.6 Influência no número de iterações internas e externas sobre a montagem do caso de teste 3 de *reads*. 46

Lista de Tabelas

1.1 Plataformas de sequenciamento 14

1.2 Montadores de primeira e segunda geração 14

3.1 Representação binária para uma solução composta por dois valores inteiros. . 28

3.2 Cálculo da roleta. 30

5.1 (Caso de teste 1) Resultado da montagem para diferentes valores do parâmetro de MAX_INTERNAS 42

5.2 (Caso de teste 1) Resultado da montagem para diferentes valores do parâmetro de MAX_EXTERNAS 42

5.3 Caso de teste 2 e influência das iterações internas no tempo e número de *contigs* obtidos pela montagem 44

5.4 Caso de teste 3 e influência das iterações internas no tempo e número de *contigs* obtidos pela montagem 44

5.5 Caso de teste 2 e influência das iterações externas no tempo e número de *contigs* obtidos pela montagem 45

5.6 Caso de teste 3 e influência das iterações externas no tempo e número de *contigs* obtidos pela montagem 45

Capítulo 1

Introdução

Em meados de 2005 surgiu uma nova geração de tecnologias de sequenciamento de DNA em larga escala que ficou conhecida como *Next-Generation Sequencing* ou simplesmente NGS ou, ainda, segunda geração de sequenciadores. Esse conjunto de tecnologias tem acelerado as pesquisas nas áreas biológicas, possibilitando a análise detalhada de organismos, sendo responsável por tornar o processo de sequenciamento mais rápido e menos custoso [1]. As tecnologias de sequenciamento anteriores eram em sua maioria baseadas no método de *Sanger* [2] e suas principais limitações são custo, tempo, velocidade e escalabilidade. Tais barreiras foram superadas pelas NGS's, que utilizam uma abordagem fundamentalmente diferente para o sequenciamento, o que desencadeou varias vantagens, como por exemplo o número de genoma humano sequenciado tem aumentado e o valor por sequenciamento tem diminuído significativamente a partir de 2005, a taxa de erro por genoma sequenciado é cada vez menor, isto é proporcionado pela qualidade da das máquinas sequenciadoras [3].

1.1 Sanger e NGS

Técnicas de sequenciamento baseadas no método *Sanger*, também ditas técnicas de primeira geração, necessitam de diversas cópias da molécula a ser sequenciada. Estas são então quebradas em fragmentos de tamanhos adequados (entre 400 - 900 pares de bases) para que assim possam ser sequenciadas diretamente. Após o processo de sequenciamento realizado pelas máquinas sequenciadoras, temos um conjunto de fragmentos chamados de *reads*, os quais devem ser montados para que a sequência original do organismo seja obtida. Em linhas gerais, o processo de montagem exige que as regiões de igualdade entre os fragmentos sejam determinadas a fim de encontrar uma ordenação entre os *reads* e assim obter a sequência original.

A Figura 1.1 ilustra as linhas gerais do processo de sequenciamento e montagem de uma molécula de DNA através do método de *Sanger*, o qual obedece as seguintes etapas:

1. **Replicação da molécula:** a molécula de DNA que se deseja sequenciar é replicada para que esteja disponível uma quantidade de material suficiente para etapa seguinte. Tal replicação acontece *in vivo*;

2. **Fragmentação:** as moléculas são quebradas em pontos aleatórios de tal forma que os fragmentos resultantes possuam tamanho adequado para serem sequenciados pelas máquinas sequenciadoras existentes no mercado;

3. **Sequenciamento:** os fragmentos são submetidos à máquina sequenciadora que geram os arquivos de cromatogramas os quais representam, por meio de gráficos, as sequências de bases que compõem os fragmentos (*reads*);

4. **Determinação das bases que compõem os *reads*:**os cromatogramas são analisados por algum *software* e a composição (sequência de bases A, C, G e T) de cada fragmento é extraída;

5. **Determinação das sobreposições entre os *reads*:** por meio da identificação de sobreposições entre os fragmentos, uma ordem entre os mesmos é estabelecida;

6. **Determinação do consenso:** o conjunto dos *reads* sobrepostos é substituído por uma sequência única, chamada consenso. O consenso pode ser determinado por simples escolha do caracter que possui maior frequência em cada uma das colunas dos *reads* sobrepostos.

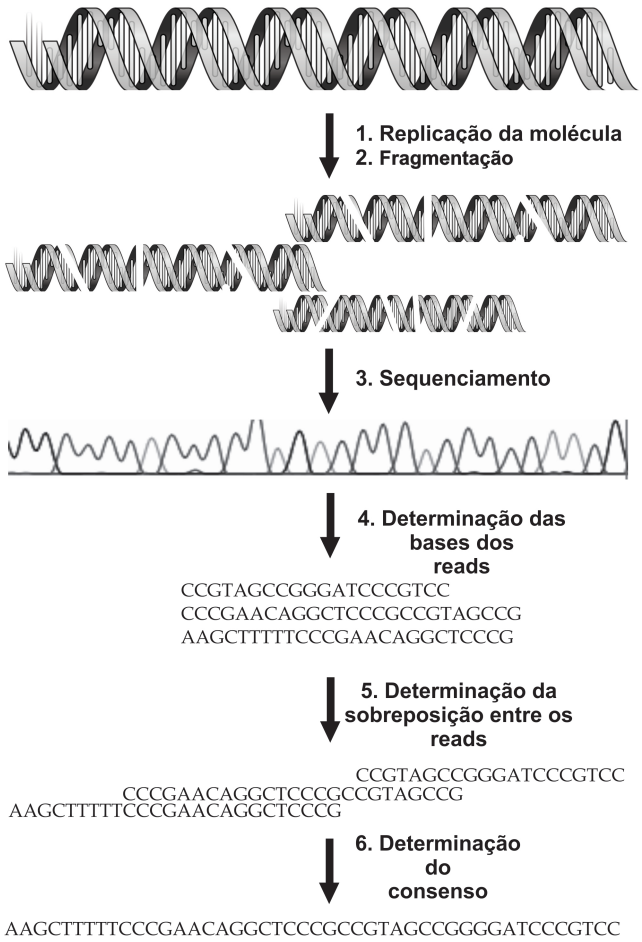


Figura 1.1: Esquema do processo de sequenciamento e montagem de *reads* adaptado de [22].

A segunda geração de tecnologias de sequenciamento tem proporcionado uma grande transformação na maneira pela qual os cientistas extraem informações genéticas de sistemas biológicos, possibilitando uma melhor visão sobre os genomas e favorecendo avanços científicos na pesquisa de doenças humanas, na agricultura e nos estudos de evolução das espécies. Estas plataformas de sequenciamentos não mais necessitam de um grande volume de DNA, sua replicação é *in vitro* o que facilita o preparo da amostra. A Figura 1.2 ilustra os principais passos de um sequenciador NGS.

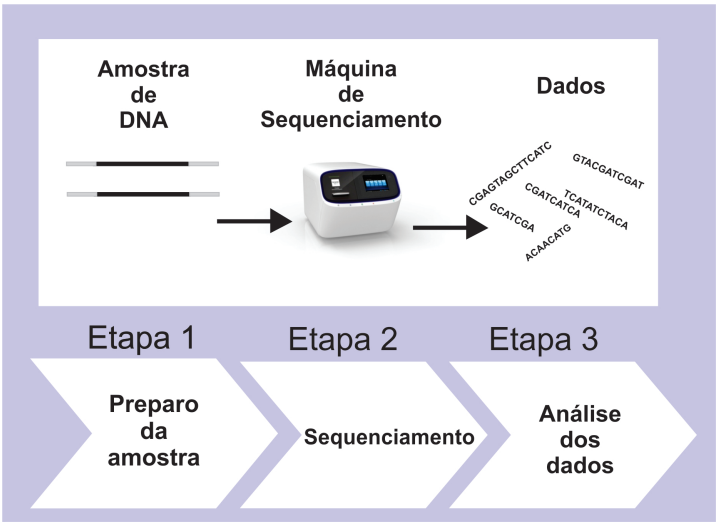


Figura 1.2: *Pipeline* do processo de sequenciamento das NGS, adaptado de [35].

As etapas do *pipeline* presente na Figura 1.2 podem ser assim descritas:

- **Preparo da amostra:** a amostra de DNA é fragmentada através de algum processo, seja ele químico, enzimático ou mecânico. Após a fragmentação, adaptadores, que são sequências de DNA artificiais conhecidas, são incorporada nas extremidade dos fragmentos. Os adaptadores utilizarão os fragmentos de DNA como modelo (*templates*) para a incorporação de bases e assim realizarem sua extensão;
- **Sequenciamento:** a máquina sequenciadora é o instrumento que executa uma série de reações químicas com o objetivo de gerar sinais que determinam a presença das bases de cada fragmento sendo estendido e assim produzem os *reads*;
- **Análise dos dados:** após a etapa de sequenciamento, temos os *reads* representados em arquivos contendo as bases correspondentes aos fragmentos. Estes precisam ser montados para que sequência do organismo alvo seja então determinada;

Existem atualmente várias plataformas de sequenciamento que são amplamente utilizadas em todo mundo. Essas plataformas de sequenciamento possuem em comum a capacidade de gerar um volume de informações muito maior que o produzido pelo sequenciamento de Sanger em tempo e custo menores. Tais plataformas de sequenciamento possuem diversas diferenças que influenciam na característica dos *reads* produzidos, como tamanho e quantidade, bem como no tempo necessário para o sequenciamento. As diferenças entre as plataformas de sequenciamento advém das técnicas de sequenciamento utilizada por cada plataforma. As

técnicas de sequenciamento utilizadas pela *454* e pela *Illumina* são ditas pirosequenciamento [4] e terminador reversível [5], respectivamente, sendo ambas baseadas em *síntese*, enquanto e tecnologia Solid é baseada em *ligação*.

A Tabela 1.1 apresenta algumas plataformas de sequenciamento utilizadas atualmente e suas principais características.

Tabela 1.1: Plataformas de sequenciamento

Plataforma de Sequenciamento	Volume de Dados	Tamanho dos Reads	Tempo de Execução
HiSeq 2500 (Illumina)	1000 GB	2 x 125 bp	48 Horas
454 GS FLX Plus System (Roche)	0.7 GB	700 bp	8 Horas
SOLiD (Life Tecnologies)	150 GB	85 bp	8 Dias
Ion Proton (Life Tecnologies)	3 GB	200 bp	2 Horas
PacBio RS (Pacific BioScience)	3 GB	3000 - 15000	20 Minutos

Juntamente com as novas tecnologias de sequenciamento surgiu a necessidade de novas ferramentas de montagem capazes de manipular um volume maior de dados com cobertura maior, bem como *reads* menores. Desde 2005 com o surgimento da NGS muitos softwares foram criados ou revisado para atenderem a essas mudanças possibilitando assim que novos genomas fossem montados. A evolução das tecnologias de sequenciamento proporcionaram dados com maior precisão, taxa de erros menores e nível de confiança maior, porém gerou a necessidade de que novos métodos fossem desenvolvidos para acompanhar tais mudanças. Duas abordagem se destacam neste cenário, grafo de sobreposição e grafo de Bruijn ambas estão descritas no Capítulo 2. A Tabela 1.2 exibe alguns montadores tanto de primeira geração (WGS) como da segunda (NGS).

Tabela 1.2: Montadores de primeira e segunda geração

Montador	Tamanho dos reads	Tecnologia	Geração	Ref
AbySS	curto	Illumina/SOLiD	WGS/NGS	[6]
Velvet	curto/longo	Sanger/Illumina/454	WGS/NGS	[7]
SOAPdenovo	curto	Illumina	NGS	[8]
Phrap	curto/longo	Sanger/Illumina/454	WGS/NGS	[8]
Ray	curto/longo	Illumina/454	NGS	[9]
Edena	curto	Illumina	NGS	[10]
Minia	curto	Illumina	NGS	[11]
Mira	curto/longo	Illumina/454 (Ion Torrent)	WGS/NGS	[12]
VCAKE	curto	Illumina	NGS	[13]
Phusion	Longo	Sanger	WGS	[14]
Geneious	curto/longo	Sanger/454/ Illumina/ Ion Torrent,	WGS/NGS	[15]
Cortex	curto/longo	454/ Illumina/ SOLID/Sanger	WGS/NGS	[16]
PASHA	curto	Illumina	WGS/NGS	[17]
Sopra	curto/longo	Sanger/Illumina	WGS/NGS	[18]
Taipan	curto	Illumina	NGS	[19]
SGA	curto/longo	Sanger/454/ Illumina/Ion Torrent	WGS/NGS	[16]

1.2 Justificativa

O problema de montagem de *reads* é um dos problemas fundamentais em Biologia Molecular, pois é o ponto de partida dos trabalhos pós sequenciamento, com o objetivo de conhecer a sequência de DNA de um indivíduo, ou de um gene, determinar mutações entre indivíduos que possuam estrutura semelhante em suas árvores filo genéticas, novos métodos de diagnósticos para doenças e vacinas entre outros. É considerado um problema de grande complexidade devido à sua natureza combinatória e ao grande volume de dados produzido pelas novas tecnologias de sequenciamento [20]. As tecnologias de sequenciamento continuam avançando e estamos rumo à terceira geração de sequenciadores que buscam resolver as limitações dos sequenciadores de segunda geração além de diminuir ainda mais o custo e o tempo do sequenciamento [21].

É muito provável que os avanços nas tecnologias de sequenciamento exijam também mudanças nos métodos e softwares de montagem de *reads*, assim como aconteceu com o surgimentos das NGS, em meados de 2005. Sendo assim, o estudo dos algoritmos existentes bem como a proposta e o desenvolvimento de novas abordagens para o problema de montagem de *reads* é uma questão que irá se manter relevante por muito tempo.

1.3 Objetivo

O objetivo geral deste projeto de pesquisa é avaliar a viabilidade de uma abordagem baseada em Algoritmo Genético para o problema de montagem de *reads*. Como objetivos específicos desta pesquisa temos: o estudo das abordagens mais utilizadas pelos softwares que realizam a montagem de *reads* proveniente das NGS; proposta e avaliação de um modelo a ser implementado por um algoritmo genético para o problema de montagem de *reads*; implementação e avaliação do modelo proposto.

1.4 Organização do Texto

O restante do texto está organizado da seguinte forma: o Capítulo 2 define o problema de montagem de *reads* e descreve as estratégias fundamentais utilizadas pela maioria dos softwares montadores de *reads*. O Capítulo 3 apresenta definições básicas necessárias ao entendimento dos Algoritmos Genéticos (AGs) bem como descreve trabalhos que se utilizam de AGs para a montagem de *reads*. O Capítulo 4 apresenta o modelo do AG desenvolvido. No Capítulo 5 os testes realizados com a implementação do modelo desenvolvido são apresentados. As considerações finais estão no Capítulo 6.

Capítulo 2

Problema de Montagem

O problema de montagem de *reads* é um dos principais problemas estudados em biologia computacional [22]. A primeira etapa de um projeto genoma é conhecer a sequência completa do organismo que está sendo estudado; a esta fase é dada o nome de sequenciamento. O DNA é uma molécula de fita dupla, sendo cada uma delas formada pela ligação de moléculas menores, os nucleotídeos. Esses diferem entre si pela presença de uma base nitrogenada, que pode ser Adenina (A), Timina (T), Citosina (C) ou Guanina (G). As fitas possuem orientações definidas como 5' $\rightarrow$ 3' na fita que recebe o nome de fita + e 3' $\rightarrow$ 5' na fita − e se unem por complementaridade entre os pares de bases A-T e C-G. A Figura 2.1 Ilustra um trecho de uma molécula de DNA.

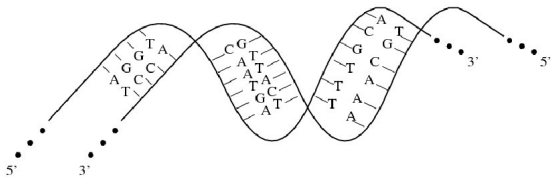


Figura 2.1: Ilustração de um trecho de molécula de DNA.

Devido ao fato das tecnologias atuais de sequenciamento não serem capazes de sequenciar um genoma inteiro de uma só vez, é necessário que diversas cópias sejam produzidas e fragmentadas, para que tais fragmentos tenham tamanhos adequados para serem sequenciados. Uma vez sequenciados os fragmentos, obtém-se um conjunto de *reads*. Cada *read* corresponde as bases de um fragmento sequenciado. Estes precisam, por fim, serem montados para que a sequência da molécula de DNA inicial seja descoberta.

A montagem dos *reads* pode ser feita através da determinação da sobreposição entre os mesmos. Considere dois *reads* $r_1 = \text{ACGCGTA}$ e $r_2 = \text{GCGTACTATG}$, de tamanhos $|r_1| = 7$ e $|r_2| = 10$. Estes podem ser sobrepostos como mostra a Figura 2.2, já que compartilham igualdade de bases em suas extremidades. Dada a sobreposição mostrada na figura, diz-se que o sufixo de tamanho 5 de r_1 é idêntico ao prefixo de tamanho 5 de r_2.

Figura 2.2: Sobreposição entre os *reads* r_1 e r_2.

Um *alinhamento* entre duas sequências s e t construídas sobre um alfabeto $\sum$ transforma s em s' e t em t', sendo s' e t' construídas sobre o alfabeto $\sum' = \sum \cup \{'-'\}$, onde $'-'$ é chamado de espaço, e $|s'| = |t'|$. Como ambas as sequências tem o mesmo tamanho, é possível avaliar a correspondência entre cada coluna do alinhamento e, assim, identificar as igualdades entre as sequências. As colunas do alinhamento que apresentam caracteres idênticos são ditas *matches*; as colunas que apresentam caracteres diferentes são ditas *mismatches*; e as colunas que apresentam um carácter com um espaço são ditas *gaps*. Diferentes tipos de alinhamentos entre duas sequências podem ser construídos, entre eles o alinhamento global, local e semi-global. Para uma revisão sobre algoritmos de alinhamentos veja [23] e [24]. A sobreposição como mostrada na Figura 2.2 pode ser vista como o alinhamento entre os *reads* r_1 e r_2, e consiste do alinhamento semi-global entre os *reads*, já que os buracos nas extremidades não são relevantes, pois no contexto do problema de montagem, busca-se por sobreposição ou alinhamentos entre sufixo e prefixo de *reads*.

Diferentes alinhamentos podem ser construídos entre dois pares de *reads* como mostrado na Figura 2.3. Entretanto, de uma maneira geral, busca-se pelo alinhamento que melhor evidencie a similaridade entre os *reads*. Sendo assim, define-se uma pontuação, ou *score* para um alinhamento, o qual pode ser calculado por um esquema que atribui 1 para *match*, -1 para *mismatch* e -2 para *gap*, por exemplo. Como, no contexto da montagem, busca-se por alinhamentos entre sufixo de um *read* e prefixo de outro *read*, os *gaps* das extremidades não são pontuados.

Figura 2.3: Alinhamentos entre os *reads* r_5 e r_6.

A escolha do alinhamento de maior pontuação (*score*) leva a determinação da ordem em que os *reads* se encontram no genoma original. Note na Figura 2.3 que temos o alinhamento 1 com $score = 1$ e o alinhamento 2 com $score = 3$, podemos então concluir que o *read* r_6 vem antes do r_5 na sequência original.

O alinhamento pode ser também definido entre um conjunto contendo $n \geq 2$ *reads*, como exemplificado na Figura 2.4. Após a construção do alinhamento, define-se como consenso, a sequência formada pela concatenação dos caracteres mais frequentes de cada coluna do alinhamento.

Figura 2.4: Sequência consenso resultante do alinhamento entre os *reads* r_1 a r_6.

2.1 Definição do Problema de Montagem de Reads

Seja G um genoma cujo sequenciamento resultou em um conjunto R de *reads*. Podemos definir o problema de montagem dos *reads* como segue:

Problema: Dado um conjunto R de *reads* originários de um genoma G, determinar a sequência de G através do conjunto R.

Entrada: Um conjunto $R = \{r_1, r_2, ..., r_m\}$ de *reads* de tamanhos variados ou não, construídos sobre o alfabeto $\sum = \{A, C, G, T\}$.

Processamento: Determinação da ordem dos elementos de R.

Saída: Uma ou mais sequências construídas sobre o alfabeto $\sum' = \sum \cup \{'-'\}$ representando os consensos das sobreposições determinadas pela ordenação entre os *reads* de R.

2.2 Considerações

Durante o processo de montagem, diversas situações devem ser consideradas, tais como erros no processo de sequenciamento, orientação desconhecida dos *reads*, regiões repetidas e baixa cobertura. Segue a descrição de tais situações dadas em [24].

- **Erros de base:** Durante o sequenciamento, bases podem ser inseridas, removidas ou ainda incorretamente detectadas. Assim, os dados produzidos pelos sequenciadores (*reads*) podem não representar fielmente os fragmentos de DNA. Muito tem se evoluído na direção da eliminação dos erros durante o processo. Como exemplo, o sistema SOLiD trabalha com uma taxa de erro de aproximadamente 0.5% por genoma sequenciado [18].

- **Orientação desconhecida:** Os fragmentos a serem sequenciados podem ser originários de ambas as fitas da molécula de DNA. Assim, após sequenciado, não se sabe se o *read* corresponde a um fragmento de DNA da fita '+' (cuja orientação é 5' → 3') ou da fita '-' (cuja orientação é 3' → 5'). Entretanto, as tecnologias de sequenciamento sempre produzem *reads* na direção 5' → 3'. Assim, para cada *read* obtido do processo de sequenciamento e não ordenado no processo de montagem, é necessário a avaliação do seu complemento reverso; fato este que pode dobrar o tamanho do conjunto de *reads* a serem processados durante a montagem.

- **Regiões repetidas:** Uma única base pode se repetir sequencialmente ao longo do DNA bem como um padrão de bases pode ocorrer diversas vezes. Tais repetições causam incertezas ao processo de montagem. Considere a sequência G e os *reads* r_i e r_j da Figura 2.5. Como determinar as posições de origem dos mesmos em G?

Figura 2.5: Ocorrência de regiões repetidas.

O *read* r_x pode se encaixar na sequência original em qualquer parte do genoma que se encontra marcado em vermelho e do mesmo modo o r_y também pode se encaixar em qualquer parte do genoma em azul.

- **Falta de cobertura:** A cobertura da posição i da sequência alvo é definida como sendo o número de fragmentos que cobrem essa posição. Após a montagem podemos calcular a cobertura média. Para isto, soma-se os comprimentos de todos os *reads* e divide-se pelo comprimento da sequência alvo montada. A cobertura incompleta acontece quando alguma posição i na sequência alvo não é coberta por nenhum *read*. Este fato é observado quando, como resultado da montagem, mais de um *contig* é encontrado, onde cada *contig* é uma sequência de DNA formada através da sobreposição maximal entre os *reads* do conjunto de entrada. A Figura 2.6 mostra um exemplo de uma cobertura incompleta.

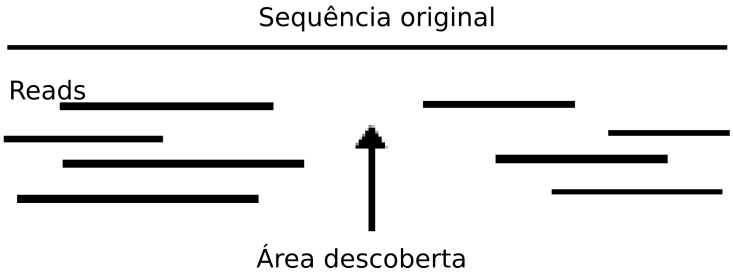


Figura 2.6: Exemplo de cobertura incompleta.

2.3 Estratégias de Montagem

Devido aos avanços das máquinas de sequenciamento e a competitividade entre as plataformas que buscam produzir melhores resultados, tornou-se essencial que os softwares montadores se adaptassem a esses novos formatos de dados produzidos como por exemplo *paired-end*. Devido a esse fato, várias estratégias foram surgindo à medida que o processo de sequenciamento foi se modificando. Nesta seção serão descrita três estratégias para o problema de montagem de *reads* que são amplamente utilizadas. São também citados alguns montadores que utilizam cada uma dessas estratégias. Em [25] e [26] são descritas detalhadamente as abordagens de Grafo de Bruijn, Grafo de sobreposição e Algoritmos Gulosos.

2.3.1 Grafo de Bruijn

O Grafo de Bruijn tem esse nome em homenagem ao matemático holandês Nicolaas Govert de Bruijn que em 1946 interessou-se em resolver o problema da supersequência circular, que consiste em encontrar a menor supersequência circular [25] que contenha todas as subsequências de tamanho k (k-mers) sobre um alfabeto β. Existem n^k k-mers em um alfabeto contendo n símbolos. Por exemplo dado um alfabeto $\beta = \{A, C, G, T\}$ e $k = 3$, temos $4^3 = 64$ trinucleotídeos, ou seja, 64 possíveis 3-mers.

Nicolaas Govert de Bruijn criou uma solução para o problema baseada em caminho Euleriano, onde é construído um grafo B em que para cada (k-1)-mer é criado um vértice. Um vértice x se conecta a outro vértice y através de uma aresta dirigida se existe uma sobreposição de tamanho $k-2$ do sufixo do vértice x com o prefixo do vértice y. As arestas do grafo de Bruijn representam todos os k-mers, um caminho Euleriano no grafo representa uma supersequência que contém exatamente cada k-mer uma única vez. A Figura 2.7 mostra um exemplo de um Grafo de Bruijn, para k = 3 e um conjunto de k-mer={ATG, TGG, GGC, GCG, CGT, GTG, TGC, GCA, CAA, AAT, ATG} proveniente de um *read* r.

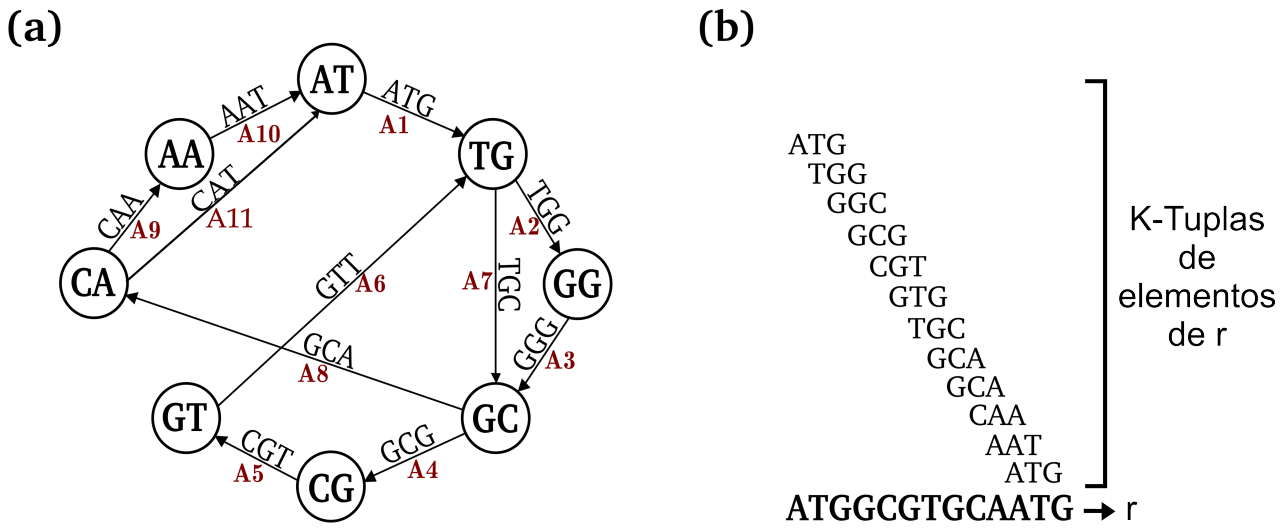


Figura 2.7: Grafo de Bruijn para o conjunto R.

Em [25] são descritos os seguintes passos para construção e busca de um grafo de Bruijn:

1. Crie um vértice para cada prefixo ou sufixo de um k-mer. Na Figura 2.7 (a) é possível visualizar que os vértices do grafo de Bruijn são (AT, TG, GG, GC, CG, GT, CA e AA). Cada vértice é inserido uma única vez no grafo.

2. Ligue, através de uma aresta dirigida, o vértice x com o vértice y se o sufixo de x é igual ao prefixo de y com tamanho $k-1$. Crie um rótulo para aresta composto pelo k-mer representado por x e y.

3. Encontre no grafo de Bruijn um caminho que passe por todas as arestas uma única vez (caminho Euleriano). Tal caminho mostra como k-mers podem ser sobrepostos. A figura 2.7 (b) mostra um exemplo de como são sobrepostos os k-mers.

Um caminho Euleriano pode ser encontrado no grafo acima pelo percurso AT → TG → GG → GC → CG → GT → TG → GC → CA → AA → AT. Através desse caminho podemos voltar a sequência alvo. Os softwares *Velvet* [7] e Abyss [6] utilizam esta estratégia. Este montadores necessitam gerenciar quantidades massivas de sequencias e geralmente não necessitam de alinhamento para formar a sequencia consenso.

2.3.2 Grafo de Sobreposição

O Grafo de Sobreposição é uma estratégia utilizada por montadores que manipulam tanto *reads* oriundo da NGS quanto *reads* oriundos de Sanger. Essa estratégia requer três etapas fundamentais:

1. **Sobreposição**: Nesta etapa, cada *read* é comparado com todos os outros *reads* do conjunto, juntamente com o seu complemento reverso. Esta comparação é o passo de

maior esforço computacional, principalmente se o conjunto de *reads* for grande. Como resultado das comparações, um grafo, chamado de grafo de sobreposição é construído, onde cada vértice representa um *read* e cada aresta dirigida do vértice x para o vértice y indica uma sobreposição do sufixo de x com o prefixo de y. Apenas sobreposições com *score* acima de um valor limite geram arestas no grafo. Na Figura 2.8 é dado um exemplo de um grafo de sobreposição para um conjunto contendo 5 *reads* sendo R1: GATCACGAA, R2: CGAAAGCAC, R3: AGATAGCGAA, R4: CGATTTAGAT e R5: AGATTACGAT.

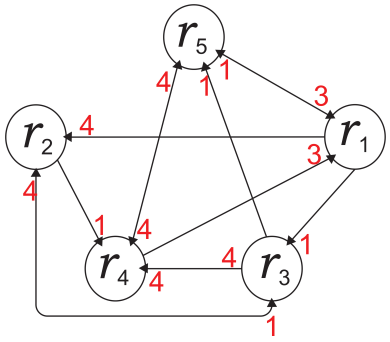


Figura 2.8: Exemplo de um grafo de sobreposição com 5 *reads*.

2. - **Layout**: Encontrar um caminho no grafo de sobreposição pode se tornar uma tarefa difícil que exige muito esforço computacional dependendo da quantidade de vértices e arestas presentes no grafo. A etapa de layout tem como objetivo diminuir o tamanho do grafo. Nesta etapa é necessário olhar para todos os caminhos hamiltoniano existentes no grafo e extrair um caminho hamiltoniano com peso máximo, ou seja o caminho que tem o maior número de sobreposição. Na figura 2.9 é dado um exemplo de layout para o grafo apresentado na figura anterior.

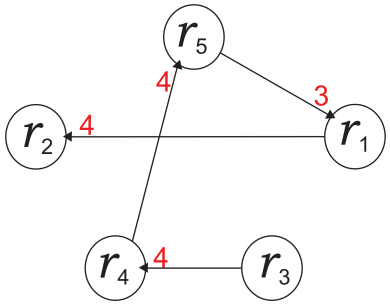


Figura 2.9: layout para o grafo de sobreposição construído na etapa anterior.

3. - **Consenso**: Na última etapa é feito o alinhamento múltiplo de todos os *reads* que compõem o caminho encontrado na etapa anterior. Na Figura 2.10 é apresentado o alinhamento múltiplo dos *reads* e mostrada a sequência consenso desse alinhamento:

Esta abordagem é utilizada por software como Celera [27] e Arachne [28].

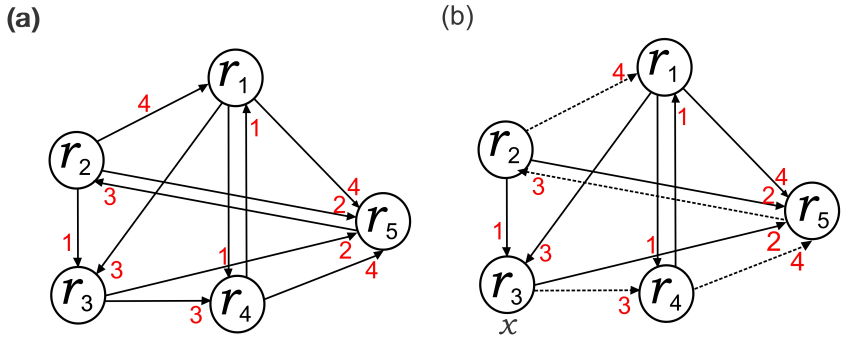
r_1 GATCACGAA
r_2 CGAAAGCAC
r_3 AGATAGCGAA
r_4 CGATTTAGAT
r_5 AGATTACGAT
AGATTACGATTTAGATGCGAAAAGCAC

Figura 2.10: Sequência consenso que representa o DNA alvo montado a partir dos *reads*.

2.3.3 Algoritmos Gulosos Baseado em Grafos

Algoritmos gulosos estão normalmente relacionados com problemas de otimização e lidam com uma sequência de passos, onde em cada passo há um conjunto de escolhas, que são realizadas baseado na que melhor contribui para se encontrar uma solução para o problema naquele instante. Um algoritmo guloso busca sempre pela melhor escolha até o momento, ou seja, ele faz uma escolha ótima para as condições locais. Eles nem sempre garantem a melhor solução como resultado, mas em muitos problemas eles são úteis e podem gerar bons resultados [29]. Tal abordagem não é amplamente utiliza no problema de montagem de *reads*, pois não busca soluções no âmbito global [30]. Montadores que se utilizam dessa abordagem trabalham com genoma pequenos.

A ideia básica de um algoritmo guloso baseado em grafos para o problema da montagem de *reads* é escolher um determinado *read* ou *contig* (conjunto de *reads* já ordenado) que é representado no grafo como um vértice x e a sua sobreposição com outro determinado vértice y é representado no grafo através de uma seta (aresta) que possui o rótulo formado através do tamanho da sobreposição entre x e y. A Figura 2.11 mostra a execução de uma abordagem gulosa baseada em grafo. Considere conjunto $R = \{r_1, r_2, r_3, r_4, r_5\}$ onde r_1 = ACTGAATGCA, r_2 = CACTAAACTG, r_3 = GCAGAGCATG, r_4 = ATGCATTGCA e r_5 = TGCAAACCAC.



(c)
r_3 GCAGAGC**ATG**
r_4 **ATG**CAT**TGCA**
r_5 **TGCA**AAC**CAC**
r_2 **CAC**TAA**ACTG**
r_1 **ACTG**AATGCA
GCAGAGCATGCATTGCAAACCACTAAACTGAATGCA

Figura 2.11: Montagem baseada em algoritmo guloso para o conjunto de *reads* R.

Na Figura 2.11 em (a) temos um grafo G que é dado como entrada. Cada vértice do grafo corresponde a um *read*. As arestas indicam a sobreposição existente entre dois *reads* e está rotulada com o tamanho da sobreposição entre os mesmos. Em (b) é apresentado o caminho que a abordagem realiza para encontrar uma montagem através do grafo G. Note que o *read* r_3 é escolhido como vértice x inicial. As arestas pontilhadas indicam o percorrido pela abordagem. Em (c) é exposto a sequência consenso formada através do caminho percorrido em (b) formando assim uma montagem. O Algoritmo 1 mostra como pode ser implementado a abordagem gulosa baseada em grafos.

Algoritmo 1: Algoritmo Guloso Baseado em Grafo.

```
Entrada: Um grafo G
Saída: sequência consenso S
inicio
    i ← 1
    Tam ← Tamanho do grafo G
    V[i] ← Escolha algum vértice em (G)
    Maior ← −∞
    enquanto (i ≤ Tam) faça
        x ← V[i]
        para (Cada vértice y adjacente x) faca
            se (Maior < ValorSobreposição(x → y)) então
                Maior ← ValorSobreposicao(x → y)
                V[i + 1] ← Vértice(G(y))
            fim
        fim
        i ← i + 1
        Maior ← −∞
    fim
    Consenso(S, V)
    Retorne (S)
fim
```

A execução do algoritmo é iniciada com a entrada de um grafo G. Um vértice x é escolhido no grafo G como vértice inicial. Durante cada iteração é encontrada a maior sobreposição entre o vértice x e seus vértice adjacentes. O vértice x é atualizado com o valor da maior sobreposição encontrado. Ao fim de todas as iterações, o procedimento `Consenso` extraí a sequência consenso, que é retornada como resultado da montagem do grafo G. Como exposto anteriormente esta abordagem não é largamente utilizadas e quando utilizada é normalmente aplicada a genoma de tamanhos pequenos. Alguns softwares utilizam esta estratégia como Phrap [31], TIGR assembler [32], Cap3 [33], SSAKE [34], SHARCGS [35] e VCAKE [13].

2.4 N50

O resultado do processo de montagem tem como saída um conjunto de *contigs*. Para avaliar o resultado produzido pelos montadores, torna-se necessário verificar a qualidade desse *contigs*. Algumas métricas são utilizadas para medir a qualidade de uma determinada

montagem, como o tamanho do maior *contig* produzido, a média e quantidades de *contigs* produzidos. O N50 é uma medida estatística que nos ajuda a compreender se a montagem foi ou não bem sucedida. O valor indica que 50% das bases da montagem estão em *contigs* maiores ou igual ao valor N50. Através do N50 é possível inferir se o conjunto de *contigs* obtido como resultado da montagem possui significado biológico. Quanto maior for o valor do N50 para o conjunto obtido maior é a significância biológica da montagem.

O valor do N50 pode ser calculado através dos seguintes passos:

1. Ordene os *contigs* resultante da montagem em ordem decrescente de seus tamanhos.
2. Some o tamanho de todos os *contigs* e divida o resultado da soma por 2.
3. Acumule a soma dos tamanhos dos *contigs* (de forma ordenada) até que esta soma seja maior ou igual ao valor encontrado no passo 2). A soma acumulada é o valor N50 a ser atribuído a montagem.

A Figura 2.12 demonstra como calcular o valor N50 para uma montagem que resultou em 6 *contigs*.

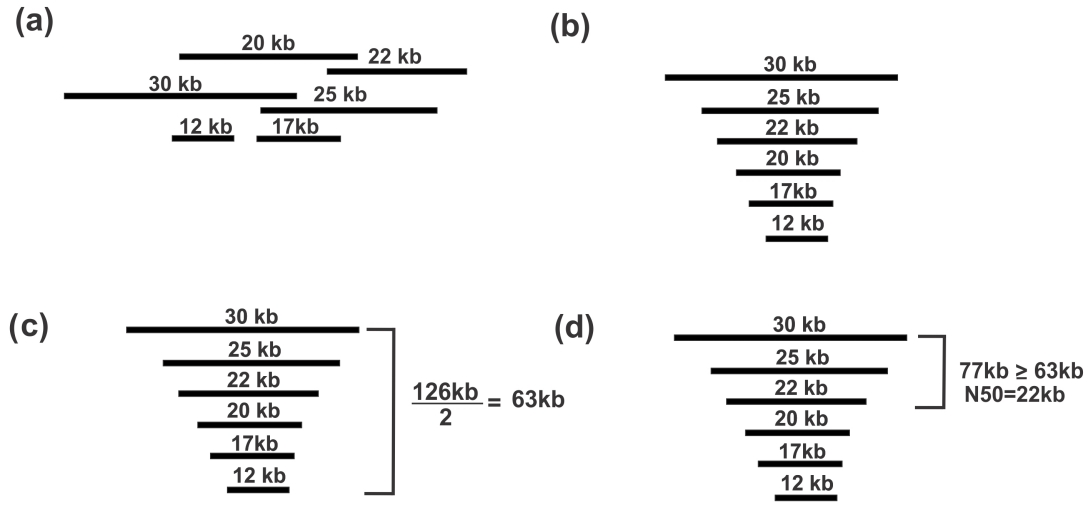


Figura 2.12: Calculo do N50 para um conjunto de *contigs*.

Capítulo 3

Algoritmo Genético

Neste capítulo os conceitos básicos de Algoritmos Genéticos são descritos. A principal referência utilizada neste capítulo foi o livro *Algoritmos Genéticos - Uma importante ferramenta da Inteligência Artificial* [36]. Ao final, trabalhos relacionados que se utilizam desta técnica para solução do problema de montagem de *reads* são brevemente descritos.

3.1 História

A inspiração para a criação dos Algoritmos Genéticos (AGs, ou GAs - *Genetic Algorithms*) surgiu através da observação da natureza, feita por Charles Darwin em uma de suas viagens. Ele observou vários animais em diferentes lugares e pôde notar que animais de uma mesma espécie eram diferentes de seus parentes em outros ecossistemas, sendo mais adaptados às oportunidades e necessidades encontradas em seus habitats. Através desta observação Darwin formulou a Teoria da Evolução, que considera que todos os indivíduos dentro de um determinado ambiente competem entre si por recursos limitados. Aqueles que não obtém recursos acabam sendo eliminados e os que obtém se tornam mais fortes. O Algoritmo Genético surgiu em 1960 com John Henry Holland [37] que ao estudar formalmente a evolução das espécies, propôs um modelo heurístico computacional com o objetivo de obter boas soluções para problemas complexos.

3.2 Conceitos e Funcionamento de um AG

Um algoritmo genético é uma técnica probabilística e não determinística para a solução de problemas de otimização. É inspirado no mecanismo de evolução das espécies e derivado dos Algoritmos Evolucionários [38]. Tais algoritmos mantêm uma população de indivíduos, os quais, ao longo das gerações evoluem de maneira similar à evolução das espécies descrita pelo Darwinismo. Em um AG, inicialmente é criada uma população aleatória de indivíduos, que representam possíveis soluções para o problema sendo tratado.

Os indivíduos da população se reproduzem gerando filhos (descendentes). Considerando que os recursos (tanto os naturais quanto os computacionais) são limitados, os pais dão lugar

aos filhos na geração seguinte da população. Assumindo que os filhos herdam as melhores características de seus pais, é esperado que os indivíduos da nova geração representem melhores soluções para o problema tratado do que os indivíduos da geração anterior. A Figura 3.1 apresenta um esquema genérico do funcionamento de um AG típico.

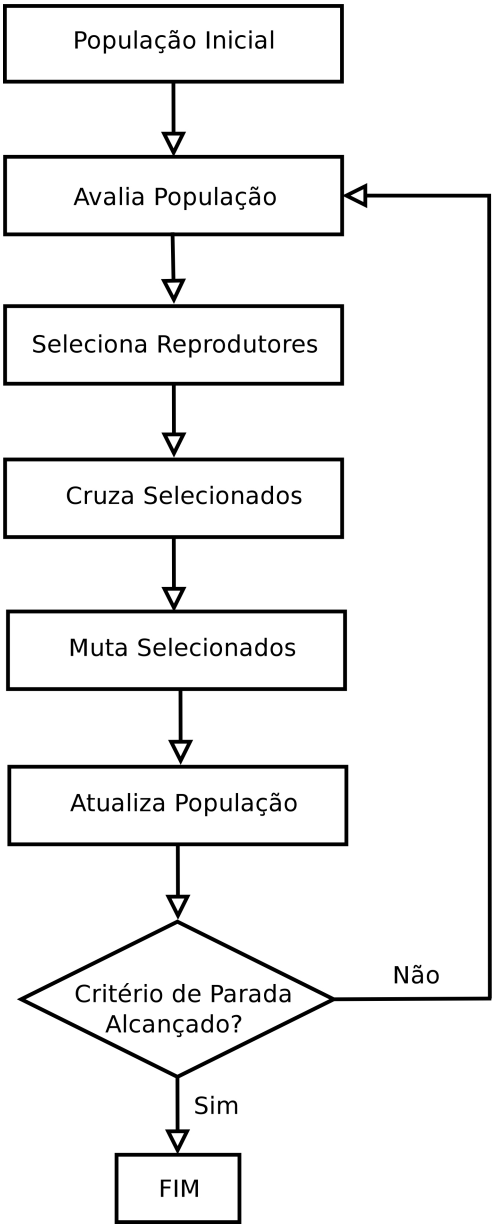


Figura 3.1: Esquema de funcionamento de um Algoritmo Genético.

Como pode ser observado do esquema da Figura 3.1, a reprodução se dá de forma sexuada, uma vez que ocorre por meio de cruzamentos entre indivíduos da população. Os indivíduos filhos, resultantes do cruzamento, podem sofrer mutações, o que aumenta a diversidade da população. Os indivíduos da população são avaliados e com base na sua contribuição como solução do problema uma pontuação é atribuída. Quanto maior a contribuição de um indivíduo, maior é a pontuação atribuída ao mesmo. Esta pontuação será deste ponto em diante denominada *fitness*.

Consideremos a existência de um AG para determinar valores de x e y capazes de maximizar a função $f(x, y) = |x * y * \sin(\frac{y\pi}{4})|$ no intervalo $[1, 15]$. Neste caso, os indivíduos devem representar valores de x e y. A representação binária é a mais usual, apesar de diversas outras representações serem possíveis como representação real, permutações, máquinas de estado finito e árvore sintática. Desta forma, valores aleatórios no intervalo são gerados e, os indivíduos são criados de forma a representar tais valores, em binário. Um indivíduo da população é também chamado de cromossomo. A Tabela 3.1 mostra dois exemplos de indivíduos gerados aleatoriamente como solução do problema de maximizar $f(x, y)$. Os primeiros 4 bits representam o valor de x e os últimos 4 bits representam o valor de y. Neste contexto, cada bit é denominada um gene do cromossomo.

Tabela 3.1: Representação binária para uma solução composta por dois valores inteiros.

Indivíduo	Valor de x	Valor de y	f_i
01000011	4	3	9,5
10011011	9	11	71,0

O valor do *fitness* do indivíduo i, representado aqui por f_i, pode ser dado pelo valor de $g(x, y) = f(x, y) + 1$. Assim, quanto maior o valor de f de um indivíduo i da população, melhor é a solução armazenada por este indivíduo em relação aos outros da população.

A evolução dos indivíduos de uma população ocorre, ao longo das gerações, devido à ação do operadores genéticos de *crossover* (cruzamento) e mutação. Por meio do operador de cruzamento, dois indivíduos denominados pais são recombinados resultando em dois novos indivíduos, chamados filhos. Diversas são as maneiras de se implementar o operador de cruzamento. A maneira mais utilizada considera a representação binária e é dita cruzamento de um ponto, uma vez que um ponto aleatório entre os genes é escolhido e as informações entre os pais são trocadas deste ponto em diante. A operação de mutação pode ocorrer nos genes dos indivíduos filhos a uma determinada taxa. No caso da representação binária, a mutação consiste na troca do valor do bit (gene) escolhido. A Figura 3.2 mostra uma recombinação entre dois indivíduos mostrados na Tabela 3.1 bem como os filhos resultantes.

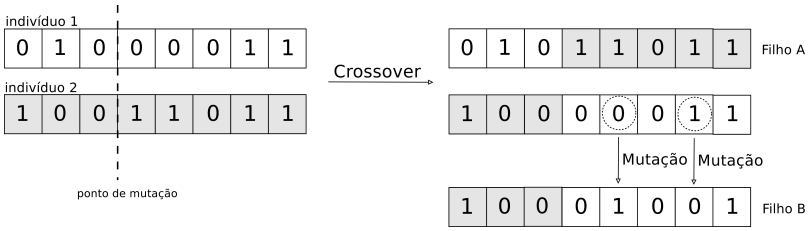


Figura 3.2: Crossover de um ponto e Mutação.

O cruzamento de dois pontos é implementado pela escolha aleatória de dois pontos entre os genes que formam os indivíduos selecionados. Os genes originários do indivíduo 1 que estão entre estes pontos serão herdados pelo Filho B, enquanto que os genes entre os pontos originários do indivíduo 2 será herdado pelo Filho A, como mostra o esquema da Figura 3.3.

Os valores correspondentes dos filhos A e B obtidos do crossover de um ponto e da mutação representados na Figura 3.2 são, $x = 5, y = 11$ e $f_A = 39,9$ e $x = 8, y = 9$ e $f_B = 51$,

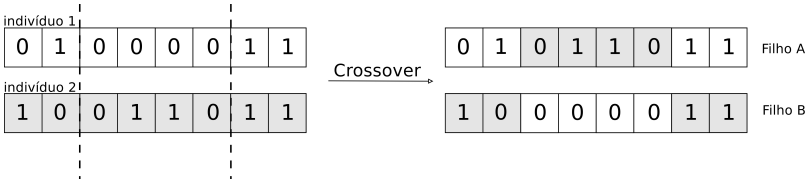


Figura 3.3: Crossover de dois pontos.

respectivamente. Tais indivíduos estarão presentes na nova geração em substituição aos pais. Apesar de termos perdido o indivíduo 2 que possui o maior fitness entre todos do conjunto de pais e filhos, o fitness médio da nova geração aumentou de 40,3 para 45,9, o que é considerado uma evolução. A evolução que ocorre durante a aplicação dos operadores genéticos deve acontecer um número determinado de vezes ao longo de T gerações. O melhor ou os melhores indivíduos existentes na última geração é considerado a solução para o problema.

Um AG é composto por algumas etapas que sempre estão presentes na solução de todos os problemas que se utilizam dessa abordagem. O Algoritmo 2 mostra os procedimentos genéricos encontrados na solução de todos os problemas que se utilizam dessa abordagem.

Algoritmo 2: Algoritmo Genético genérico.

$t \leftarrow 0$;
`Inicializacao(`P, t`);`
enquanto $(t \neq T)$ **faça**
  `Avaliacao(`P, t`);`
  `Selecao(`P', P, t`);`
  `Cruzamento(`P', t`);`
  `Mutacao(`P', t`);`
  `Atualizacao` (P', P, t);
fim
Retorne `Melhor(`P`);`

Um Algoritmo Genético tem seu início a partir do procedimento de `inicialização`. Tal procedimento consiste na criação de uma população inicial, sobre a qual o AG irá ser executado. Em geral, pode-se dar início a uma população de maneira aleatória, ou seja os indivíduos são gerados de forma randômica com objetivo de aumentar sua diversidade genética garantindo, desta forma, um maior alcance do espaço de busca. Após a criação da população inicial, é necessário avaliar todos os indivíduos gerados. Desta forma, o próximo procedimento executado é o de `avaliação`. A avaliação define o quanto um indivíduo contribui para a solução final do problema. A escolha dos indivíduos que irão se reproduzir é baseada no *fitness* fornecido pelo procedimento de avaliação. Indivíduos de maior *fitness* são os mais aptos. Entretanto, os indivíduos mais fracos (de menor *fitness*) podem também ser selecionado, porém com uma chance menor de serem escolhidos para o processo de reprodução e assim gerarem novos filhos diversificando a população. O procedimento responsável por selecionar os indivíduos da população é denominado `seleção`. A seleção dos pais é implementada tradicionalmente por métodos denominados seleção por torneio, seleção por *ranking* e seleção seleção por roleta, porém existem diversos tipos de seleção. Todos estes métodos favorecem a seleção dos indivíduos de maior *fitness*

Para exemplificar o método de seleção por roleta, considere os indivíduos da Tabela 3.2 para os quais o tamanho da fatia da roleta bem como o correspondente em graus está calculado para cada indivíduo. O cálculo da percentagem da roleta dada a um indivíduo i, para $1 \leq i \leq 6$, é realizado pela expressão $(f_i / \sum_{i=1}^{6} f_i) * 100$ que calcula o valor de percentagem e $(p_i * /100)$ para calcular o valor em graus, tais valores que refletem o quão representativo é o indivíduo na população.

Tabela 3.2: Cálculo da roleta.

Indivíduo	Valor de x	Valor de y	f_i	p_i % da rotela	Graus da roleta
10001001	8	9	51,2	31,3	111,6
01001010	4	10	41,0	25,0	90,0
01011011	5	11	39,9	24,3	87,4
11010010	13	2	27,0	16,4	59,0
01000001	4	1	3,8	2,4	8,6
10001000	8	8	1,0	0,6	2,8
		Somatório = $\sum$	163,9	100	360

A roleta resultante dos cálculos apresentados na Tabela 3.2 é apresentada na Figura 3.4. Uma vez construída a roleta, sorteia-se um valor inteiro entre 0 e 164,6 para que a seleção de um indivíduo aconteça. Como exemplo, se o valor sorteado é igual a 66, o terceiro indivíduo da tabela (01011011) é selecionado.

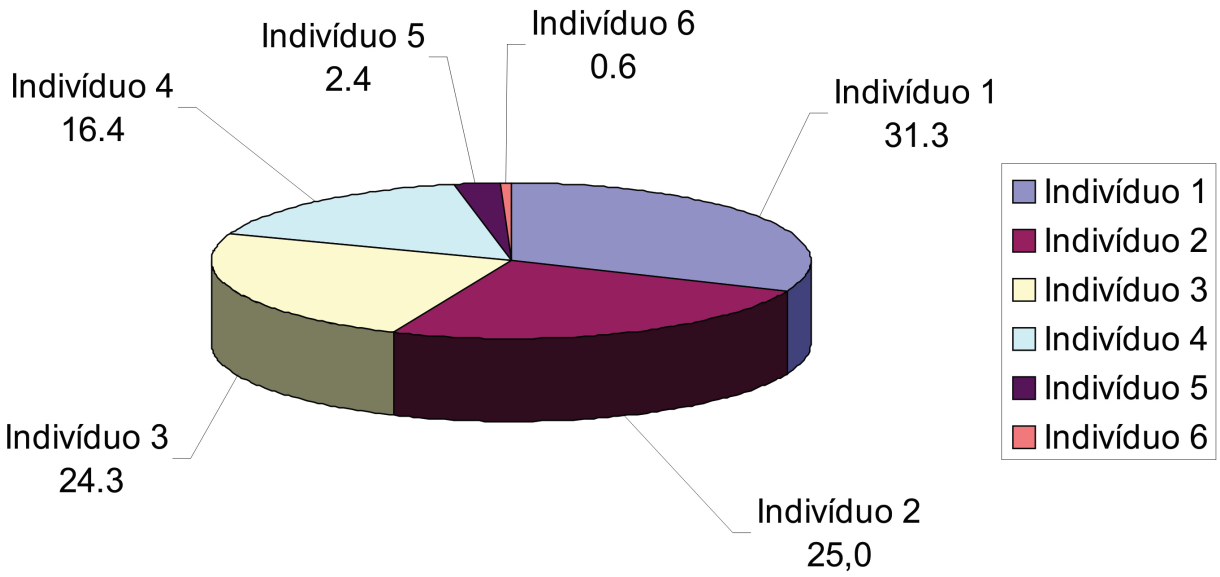


Figura 3.4: Exemplo da seleção por roleta para a Tabela 3.2.

Os procedimento de `cruzamento` e `mutação` são operadores genéticos e foram descritos na seção acima. O objetivo desses procedimento é garantir que o espaço de busca

seja explorado e assim boas soluções sejam encontradas. No procedimento denominado `atualização` a população antiga é substituída por uma nova população, formada pelo cruzamento dos indivíduos selecionados da população anterior. As formas mais conhecidas de atualização são $(x+y)$ e (x, y), também chamadas de estratégia soma e estratégia vírgula. Na estratégia soma, indivíduos da população anterior convivem com a população formada por seus filhos. Essa estratégia também pode ser chamada de elitismo e geralmente uma percentagem muito pequena é selecionada para a próxima geração, pois corre-se o risco de uma convergência prematura do AG, podendo gerar assim diversidade. Na estratégia vírgula a população anterior não convive com a próxima população, perdendo-se então soluções boas encontradas. Por fim o procedimento `finalização` é o responsável por determinar se a execução do AG (evolução de população) será concluída ou não. Tal ação é realizada a partir da execução de testes baseados em uma condição de parada pré-estabelecida. Tal condição de parada pode variar desde a quantidade de gerações desenvolvidas até o grau de proximidade dos valores de *fitness* obtidos.

Uma das vantagens em se utilizar um AG é o alto grau de paralelismo que pode ser utilizado em sua execução. Os indivíduos de uma população podem ser avaliados de forma independente, isso permite que o processo de avaliação seja executado em máquinas diferentes.

3.3 Trabalhos Relacionados

Alguns trabalhos que utilizam AG para solucionar o problema de montagem de *reads* são encontrados na literatura. O mais recentes deles data de 2005. Em *A Genetic Algorithm Approach to Solving DNA Fragment Assembly Problem*, descrito em [22] é proposto um algoritmo genético para o problema de montagem de *reads*. O modelo considera que o tamanho da sequência a ser montada é conhecido e que durante o processo de sequenciamento não ocorrem erros. Esse cenário é apenas hipotético, tendo em vista que quase sempre o tamanho da sequência a ser montada é desconhecido e que durante o processo de sequenciamento ocorrem erros assim como especificado no Capítulo 2. Essa abordagem procura maximizar as sobreposições existentes no conjunto de *reads*. Neste trabalho não está especificado claramente a representação dos indivíduos, porém sabe-se que eles são compostos por uma permutação aleatória de inteiros, onde cada número inteiro representa um *read* do conjunto de entrada. Alguns operadores genéticos foram utilizados, entre eles os operadores de cruzamento e mutação. A avaliação é feita através de uma função de *fitness* que busca as maiores sobreposições entre todos os indivíduos da população. A eficiência da abordagem foi demonstrada através de experimentos considerando *reads* de até 100 bp. Porém existe possibilidade de que o modelo seja eficiente para *reads* de tamanho maior e sequências com tamanho desconhecido. Os resultados obtidos mostram que o AG desenvolvido se comportou de forma promissora para pequenos e médios conjuntos de *reads* e que futuramente pode ser empregado em sequências de maior número de bases e quantidade de *reads*.

Em *Metaheuristics for the DNA Fragment Assembly Problem* descrito em [20], é relatado um estudo descrevendo o problema de montagem de *reads* e suas complicações. O trabalho apresenta quatro heurísticas distintas para o problema de montagem de *reads*: um método *CHC (Cross generational elitist selection, Heterogeneous recombination, and Cataclysmic mutation)*, um método *SS (Scatter Search)*, um método utilizando um Algoritmo Genético e

um método baseado em *SA (Simulated Annealing)*. Um breve descrição de cada método foi apresentada. A população é representada por indivíduos contendo um número fixo de genes formado por uma permutação aleatória de números inteiros, onde cada número inteiro representa um *read* do conjunto de entrada. O objetivo do trabalho é comparar a solução produzida por cada heurística. Os testes foram realizados utilizando dois conjunto de *reads* reais disponíveis na base do NCBI. As comparações efetuadas entre os quatro métodos mostraram que o *Simulated Annealing* produz melhores resultados que as demais heurísticas. O algoritmo genético ficou com o terceiro melhor resultado, porém sua performance pode ser melhorada através de um número maior de gerações.

Em *Genetic Algorithms for DNA Sequence Assembly* [39], é descrito um algoritmo genético aplicado ao problema de montagem de *reads*. Uma comparação entre o problema de montagem e o caixeiro viajante é apresentada, mostrando que existem pontos em comuns entre os dois problemas. A representação de um indivíduo na população é feita de forma binária. Duas funções de *fitness* foram utilizadas. O trabalho concluiu que o algoritmo genético é um método promissor para alcançar soluções utilizáveis de forma rápida, porém as funções de *fitness* responsáveis por avaliar os indivíduos possuem falhas e permitem que indivíduos de qualidade ruim sejam integrado a solução final.

Em *Parallel GAs in Bioinformatics: Assembling DNA Fragments* [40] é apresentado um algoritmo genético para o problema de montagem de *reads* em paralelo. Este trabalho mostra o problema teórico bem como as dificuldades encontradas durante sua manipulação. Foi desenvolvido e detalhado as principais características do montador denominado PGA para que fosse atacado o problema de montagem de *reads*. Como critério de avaliação foi medida a melhor pontuação obtida, a média de pontuação obtida, a média de avaliação realizada e a média de tempo gasto durante a execução do PGA. Está também especificado como realizar o paralelismo em um AG abordando o problema de montagem de *reads*.

Capítulo 4

Algoritmos Genéticos e a Montagem de Reads

Neste capítulo é apresentado o modelo proposto para o problema de montagem de *reads* utilizando uma abordagem via Algoritmo Genético (GA). A forma como um indivíduo é representado na população e o método pelo qual um indivíduo é avaliado, bem como os operadores genéticos utilizados neste modelo são especificados ao decorrer deste capítulo. O modelo descrito foi implementado e os resultado obtidos nos testes realizados são apresentados no Capítulo 5.

4.1 Representação de um Indivíduo

Um dos fatores fundamentais para o sucesso na utilização de um AG é a escolha da representação adequada de sua população para o problema a ser abordado, bem como os parâmetros utilizados na configuração de sua execução. A forma como um indivíduo é representado dentro de uma população e a interpretação das informações que compõem cada indivíduo influenciam diretamente na eficiência e tempo de execução.

Seja o problema de montagem de *reads* como definido na Seção 2.1. O modelo (ideal) de cromossomo para tratar este problema deveria conter k genes, sendo que cada gene armazena um *read* ou apenas o seu identificador. A Figura 4.1 mostra um indivíduo representado por $k = 16$ genes.

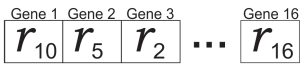


Figura 4.1: Representação de um indivíduo composto por $k = 16$ genes.

Um bom indivíduo da população possui genes adjacentes g_i e g_j, para $1 \leq i \leq k-1$ e $j = i + 1$ que compartilham sobreposições entre um sufixo do gene i e um prefixo do gene j de tal forma que a sequência consenso da união de todas as sobreposições entre os genes consecutivos de um indivíduo representa a montagem do genoma G. É possível que o tamanho do genoma G a ser montado seja desconhecido. Mesmo em casos onde o tamanho

de G é previamente conhecido a definição do valor do parâmetro k não é uma tarefa trivial. Considere o genoma *Escherichia coli* [41], cujo o tamanho é aproximadamente 4,641,652 milhões de pares de base, e o sequenciamento (ERX539052) resultou em 1,680,150 milhões de *reads*. Como neste exemplo não se sabe o tamanho da cobertura entre os *reads*, somado ao fato de diversos *reads* cobrir uma mesma região do genoma, não é possível calcular o valor de k. Sendo assim , o modelo proposto considera $k = 2$. Para que um indivíduo seja bem avaliado na população, é preciso que exista uma sobreposição entre um sufixo do gene 1 com o prefixo do gene 2, ou que a sequência de um gene esteja contido inteiramente na sequência do outro. Quanto maior for a sobreposição entre dois genes, melhor será sua avaliação. A Figura 4.2 mostra o exemplo de um indivíduo utilizado nesse modelo.

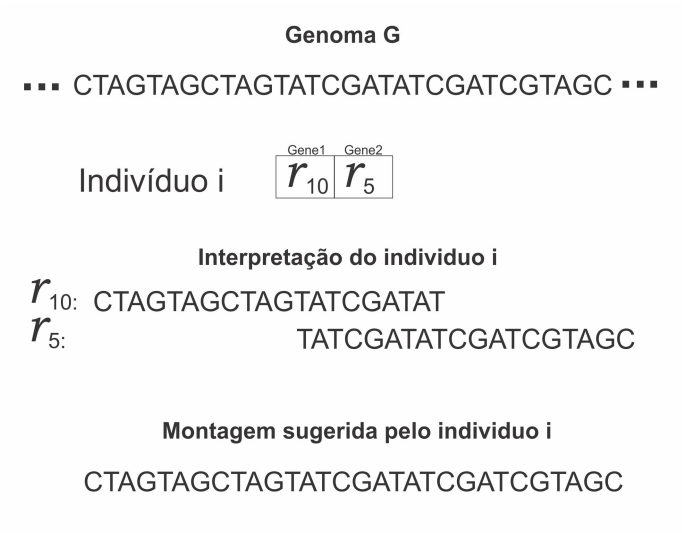


Figura 4.2: Representação e interpretação do indivíduo no modelo proposto.

É fato que indivíduos de tamanho $k = 2$ não permitem a montagem de um genoma. A proposta é que o resultando da montagem seja novamente considerado como um *read* e montado novamente em uma próxima iteração do algoritmo.

4.2 Avaliação

A avaliação de um indivíduo pertencente a uma população P é uma das principais tarefas a ser executada nessa abordagem. O processo de avaliação é de fundamental importância para definir quais indivíduos possuem conteúdo representativo de montagem. Indivíduos cujos genes adjacentes possuem pequena sobreposição devem ser desconsiderados. O método proposto neste modelo para avaliação de cada indivíduo é baseado na ocorrência de *k-mer*. Um *k-mer* consiste na sequência das k últimas bases do fragmento de DNA representado pelo gene 1.

Após extraído o *k-mer*, procura-se a ocorrência do mesmo no segundo gene. Caso o *k-mer* seja encontrado no gene 2 torna-se necessário calcular a pontuação da sobreposição entre o primeiro e o segundo gene nas posições anteriores a ocorrência do *k-mer* como na Figura 4.2. Uma tolerância de no máximo 3 *mismatch* é permitida na sobreposição. Ao encontrar um indivíduo que satisfaça a estas condições, os *reads* representado pelos genes do indivíduo são

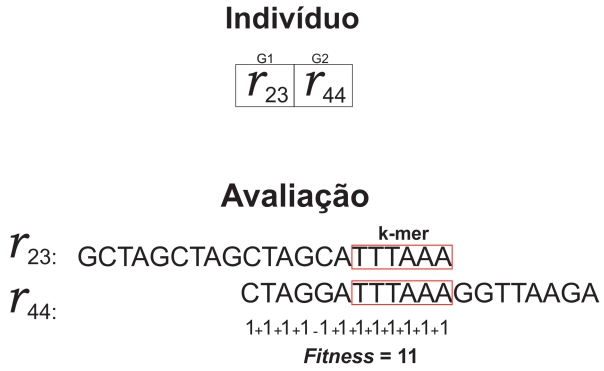


Figura 4.3: Processo de avaliação de um indivíduo.

removidos do conjunto de *reads* a serem montados e a um novo *read* contendo a sequência consenso entre os dois *reads* removidos é inserido no conjunto o qual será considerado em uma nova iteração do AG.

Durante a avaliação dos indivíduos um *read* pode estar contido dentro de outro, visto que devido a estratégia de montagem, o tamanho dos *reads* podem diferir bastante. A Figura 4.4 mostra uma situação em que não é possível realizar a montagem dos fragmentos de um indivíduo, pois o gene 2 está contido no gene 1.

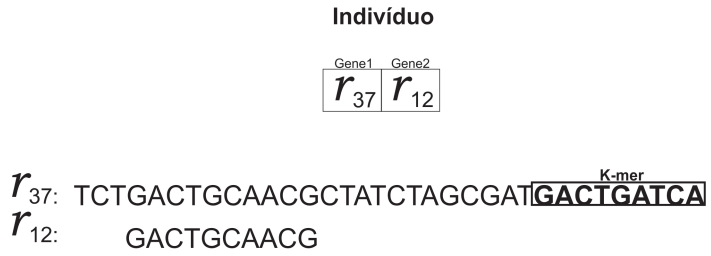


Figura 4.4: Exemplo de *reads* contidos .

Essa situação deve ser considerada, pois influencia diretamente no número de *contigs* produzido por este modelo. Foi implementado o algoritmo de KMP (*Knuth–Morris–Pratt*) para que ao fim todas as iterações do AG, *reads* menores sejam avaliados para determinar se estes estão contidos em *reads* maiores e assim sejam descartados.

4.3 Operadores Genéticos

Os operadores genéticos tem por objetivo diversificar a população, possibilitando a criação e avaliação de novos indivíduos. O operador de cruzamento é aplicado a pares de indivíduos da população. Tais indivíduos são selecionados aleatoriamente, ou seja, todos os indivíduos possuem a mesma probabilidade de serem selecionados. Logo após o processo de seleção os pares são combinados, dando origem a dois novos indivíduos. A troca do material genético é feita substituindo o gene 2 de um indivíduo pelo gene 2 do outro indivíduo. A Figura 4.5 mostra como o operador de *crossover* funciona.

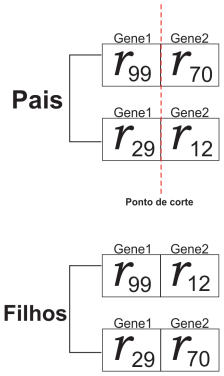


Figura 4.5: Operador de *crossover* sendo aplicado em um par de indivíduos.

A mutação é aplicada com uma probabilidade menor aos indivíduos filhos resultantes do operador de cruzamento. A mutação consiste em alterar um indivíduo trocando o conteúdo do primeiro gene com o conteúdo do segundo. A Figura 4.6 mostra como a mutação opera sobre um indivíduo.

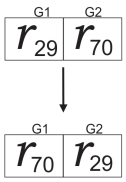


Figura 4.6: Operador de mutação sendo aplicado em um indivíduo .

Note que por meio do operador de mutação, um indivíduo que contêm *reads* consecutivos no genoma armazenados na ordem inversa pode ser beneficiado. A Figura 4.7 dá um exemplo desta situação.

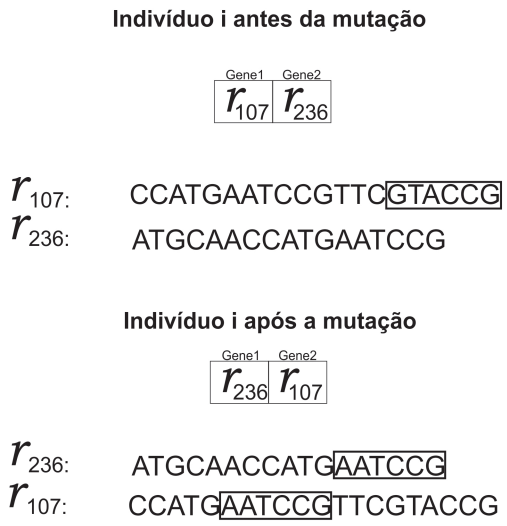


Figura 4.7: Exemplo da mutação promovendo um novo indivíduo.

Após a aplicação dos operadores genéticos é necessário avaliar os indivíduo resultantes (filhos) com os indivíduos progenitores (pais). A atualização da população é feita comparando o *fitness* dos pais com os filhos, caso algum filho possua um *fitness* melhor que o de seu pai, ele é inserido na população ocupando o lugar de seu pai.

4.4 Algoritmo

O algoritmo referente ao modelo descrito neste Capítulo é apresentado a seguir. As variáveis MAX_EXTERNAS(iterações externas) e MAX_INTERNA(iterações internas) definem o número máximo de iterações externas e internas a serem executadas.

Algoritmo 3: Montagem de *reads* via AG.

```
Entrada: Um Conjunto de reads a ser montado
Saída: Um conjunto de reads montado
inicio
    R ← LeiaReads(input.fa)
    Externas ← 0
    enquanto (Externas < MAX_EXTERNAS) e (número de elementos em
    R!=0) faça
        P ← Inicializa(R)
        Internas ← 0
        tp ← número de elementos de P
        enquanto (Internas < MAX_INTERNAS) e (tamanho de P > 1/3*tp)
        faça
            Avaliação(P)
            P' ← Seleção(P)
            cruzamento(P')
            Mutação(P')
            Avaliação(P')
            P ← Atualização(P')
            Interna ← Interna + 1
        fim
        R ← Insere(P)
        Externa ← Externa + 1
    fim
    R ← KMP(R)
fim
```

A função `Inicializa` seleciona *reads* e constrói os indivíduos da população de maneira aleatória. A função `Avaliação`, determina o *fitness* de cada indivíduo da população por meio da determinação da sobreposição entre os genes de um organismo. Os indivíduos considerados boas soluções para o problema são removidos da população logo após a avaliação, uma vez que estes não precisam evoluir pois já representam boas soluções. Os indivíduos a serem submetidos ao operador de cruzamento são selecionados aleatoriamente com a mesma probabilidade através da função `Seleção`. Os indivíduos selecionados compõem a população P'. A população P' é dividida em pares e a função `cruzamento` é aplicada a estes pares. Logo após o operador de cruzamento, a função de `mutação` é aplicada. Uma nova seleção é realizada considerando somente os indivíduos que passaram pelo cruzamento. Após os operadores genéticos serem aplicados, os indivíduos filhos resultantes são avaliados e caso representem uma boa solução substituirão os pais na população P.

Note que durante as execuções do laço mais interno ocorrem as tentativas de montagem de um subconjunto de *reads*. Ao final destas tentativas, cujo número máximo é

MAX_INTERNA, o consenso entre os *reads* correspondentes no conjunto R, diminui o tamanho deste conjunto e os *reads* não montados retornaram ao conjunto R.

O laço mais externo, que é executado no máximo MAX_EXTERNAS vezes é necessário para vencer a limitação do tamanho do indivíduo(k=2) proposto pelo modelo. A cada iteração externa, o tamanho do consenso dos bons indivíduos obtidos nas iterações mais internas tende a aumentar.

Capítulo 5

Implementação, Testes e Resultados

Neste capitulo é descrita as linhas gerais da implementação do modelo descrito no Capítulo 4, bem como os testes que foram realizados e os resultados obtidos. O objetivo deste capítulo é demonstrar a viabilidade do modelo e os desafios na sua utilização. Foi estudada e utilizada uma ferramenta de simulação de *reads* oriundos das novas tecnologias de sequenciamento para produzir os casos de teste.

5.1 Sobre a Implementação do Modelo

A implementação do modelo do AG descrito no Capitulo 4 foi desenvolvida na linguagem de programação C. Para o ambiente de desenvolvimento foi utilizado o Sistema Operacional Linux 64 bits. Como ferramenta de desenvolvimento foi utilizado a IDE Code Blocks disponibilizada em `http://www.codeblocks.org/`.

A Figura 5.1 mostra o fluxo e as principais funções implementadas no modelo.

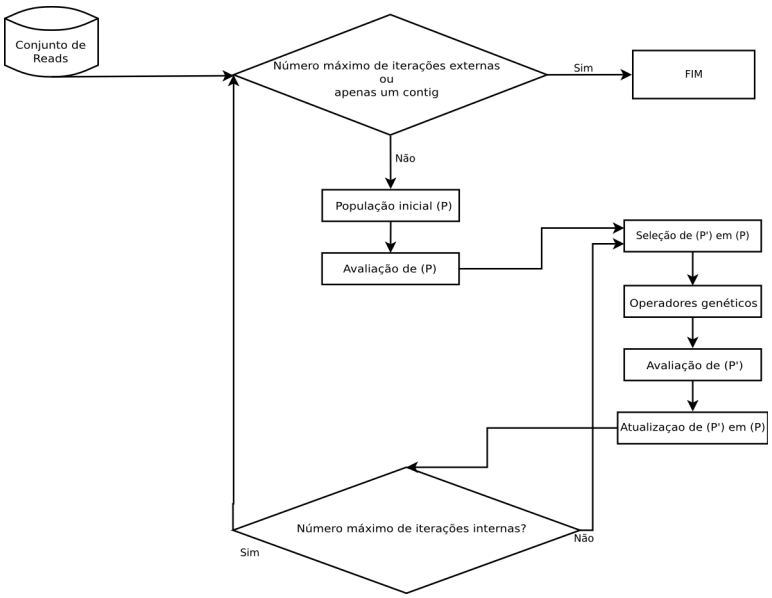


Figura 5.1: Diagrama de fluxo do modelo.

5.2 Sobre as Ferramentas

Nesta seção são apresentadas duas ferramentas que foram estudadas. O ART e o *Velvet* são ferramentas utilizadas em vários trabalhos no meio acadêmico. Sua utilização é motivada pelo fácil acesso às suas versões atuais e por serem *open source*.

5.2.1 ART

O ART [42] é uma ferramenta cujo objetivo é simular a produção de *reads* originários de sequenciadores da nova geração. O ART é amplamente utilizado para criar conjunto de *reads* a serem utilizados nos testes preliminares de novas ferramentas de montagem de *reads*. Essa ferramenta permite controlar a taxa de erro e a qualidade sobre o conjunto de *reads* a ser gerado. Isso é feito através de parâmetros que são fornecido ao ser iniciar a sua execução. Atualmente o ART é capaz de gerar *reads* baseados nas três maiores plataformas de sequenciamento, *Roche*, *Illumina* e *SOLiD*.

5.3 Testes e Resultados

Nesta seção estão apresentados os testes e resultados obtidos através do modelo desenvolvido e implementado. Os testes demonstram qual o comportamento do AG mediante as configurações de parâmetros estabelecidas, bem como da característica do conjunto de *reads* a ser montado. Devido ao fato do AG ser uma técnica probabilística e não determinística, a cada execução resultados melhores ou piores podem surgir, entretanto sabe-se que o AG sempre converge para boas soluções [44].

Cada teste foi executado 10 vezes e uma média de *contigs* obtidos foi calculada. A máquina utilizada para a execução dos testes possui um processador *Intel Xeon* E5-2620 2.00 GHz com 64GB de memória RAM utilizando o sistema operacional Linux 64 *bits*.

Os arquivos de entradas foram simulados através do ART, sendo os *reads* produzidos conforme a plataforma *Illumina*. Como refêrencia foram utilizadas as 2610 primeiras bases do genoma da bactéria *Escherichia coli* [41].

O primeiro caso de teste (C1) é composto por 100 *reads*, cada um com comprimento igual a 35 pares de bases e de sobreposição entre dois *reads* consecutivos na sequência de referência de 10 pares de bases. A Figura 5.2 mostra como os *reads* estão dispostos em relação a referência.

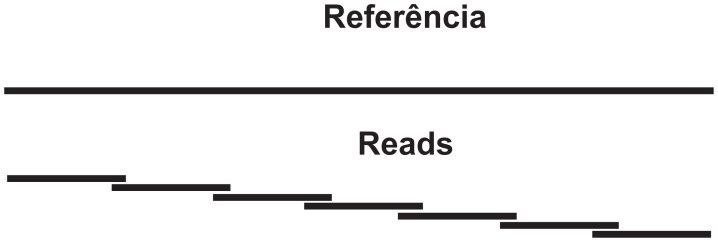


Figura 5.2: *Layout* do conjunto de teste 1.

O layout do conjunto de teste esquematizado na Figura 5.2 é hipotético e foi criado com objetivo de verificar se o modelo é capaz de chegar à montagem de um único *contig*, ou seja, se o AG converge para a solução ótima. As Tabelas 5.1 e 5.2 mostram os resultados obtidos durante os testes realizado para o primeiro caso de teste. Na Tabela 5.1 o número de iterações externas (parâmetro MAX_EXTERNAS do Algoritmo 3) está fixo em 200 e o número de iterações internas (parâmetro MAX_INTERNAS do Algoritmo 3) sofre variações. Através dos resultados é possível perceber que conforme o número de iterações internas cresce o número de *contigs* tende a diminuir e o tamanho dos *contigs* tende a aumentar, o que indica que a montagem esta acontecendo.

Tabela 5.1: (Caso de teste 1) Resultado da montagem para diferentes valores do parâmetro de MAX_INTERNAS

N° de Iterações Internas (MAX_INTERNAS)	**N° de *Contigs***	**Maior *Contig* (bp)**	**Tempo (min:seg)**
10.000	72	317	01:30
20.000	63	352	02:35
30.000	54	400	03:12
40.000	45	612	04:15
50.000	34	843	05:28
60.000	27	920	06:10
70.000	22	1210	06:54
80.000	15	1540	07:24
90.000	7	1912	08:03
100.000	1	2610	08:42

Na Tabela 5.2 o número de iterações internas está fixo em 100.000 e o número de iterações externas está sendo variado. Novamente é possível perceber que a quantidade de *contigs* está diminuindo e o tamanho dos *contigs* aumentando.

Tabela 5.2: (Caso de teste 1) Resultado da montagem para diferentes valores do parâmetro de MAX_EXTERNAS

N° de Iterações Externas (MAX_EXTERNAS)	**N° de *Contigs***	**Maior *Contig* (bp)**	**Tempo (min:seg)**
10	87	218	00:30
20	78	296	01:11
30	71	331	01:44
40	64	448	02:03
50	52	509	02:31
100	22	1512	05:22
150	8	2112	07:05
200	1	2610	08:42

Perceba que conforme o número de iterações internas e externa aumentam é possível diminuir o número de *contigs* e aumentar o seu tamanho. A Figura 5.3, mostra graficamente

os resultados apresentados na Tabela 5.1 e 5.2 e como esses parâmetros influenciam na montagem. Durante os testes realizados com o primeiro caso de teste foi feita uma verificação analisando se cada *contig* estava correspondo com a sequência de referência. Isso torna-se necessário para validar se o processo de montagem esta obtendo *contigs* verdadeiros.

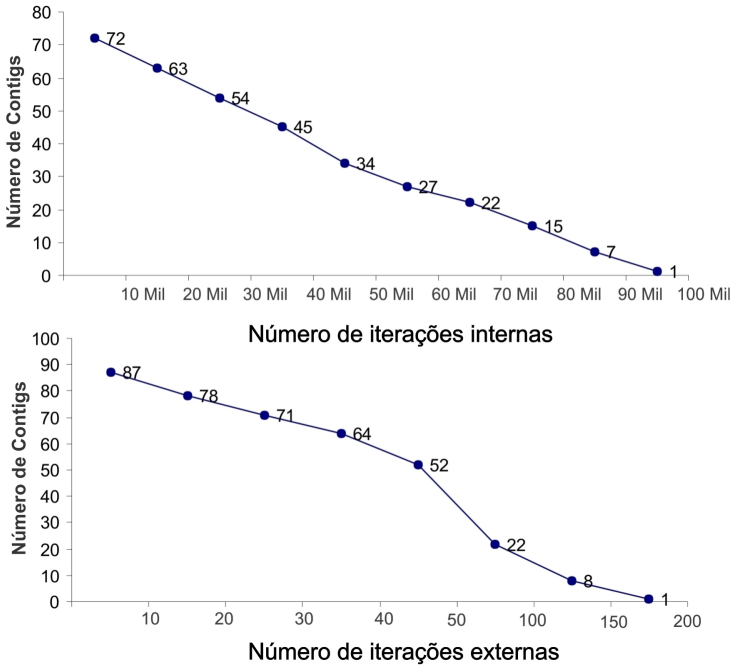


Figura 5.3: Influência no número de iterações internas e externas sobre a montagem de *reads*.

O segundo e o terceiro caso de teste é composto por respectivos 300 e 400 *reads* contendo 35 pares de bases cada. O *layout* referente a estes dois testes é esquematizado através da Figura 5.4.

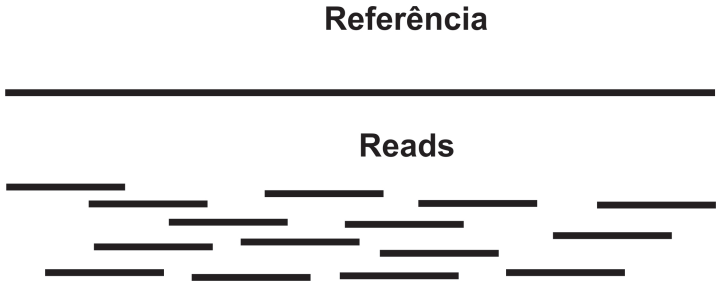


Figura 5.4: Disposição dos reads do segundo e terceiro caso de teste.

Uma referência contendo 1500 e 1700 pares de bases foi utilizada para confecionar o segundo e o terceiro caso de teste. Os resultados obtidos durante os testes estão apresentados na tabela 5.3 e 5.4 para o número de iterações internas e 5.5 e 5.6 para o número de iterações externas.

Tabela 5.3: Caso de teste 2 e influência das iterações internas no tempo e número de *contigs* obtidos pela montagem

Número de Iterações Internas (MAX_INTERNAS)	**Caso de Teste 2**		
	Número Contigs	**Maior Contig**	**Tempo (min:seg)**
10.000	236	104	01:05
20.000	212	92	02:40
30.000	192	113	03:32
40.000	158	117	04:23
50.000	145	123	05:55
60.000	124	143	06:44
70.000	108	156	07:20
80.000	91	194	08:35
90.000	64	236	09:40
100.000	52	315	10:20

Tabela 5.4: Caso de teste 3 e influência das iterações internas no tempo e número de *contigs* obtidos pela montagem

Número de Iterações Internas (MAX_INTERNAS)	**Caso de Teste 3**		
	Número Contigs	**Maior Contig**	**Tempo (min:seg)**
10.000	341	80	01:30
20.000	322	81	02:57
30.000	293	90	04:31
40.000	263	96	06:00
50.000	240	103	07:45
60.000	205	145	09:10
70.000	181	171	11:20
80.000	163	102	13:15
90.000	137	254	14:07
100.000	105	296	

Tabela 5.5: Caso de teste 2 e influência das iterações externas no tempo e número de *contigs* obtidos pela montagem

Número de Iterações Internas (MAX_EXTERNAS)	Caso de Teste 2		
	Número Contigs	Maior Contig	Tempo (min:seg)
10	281	65	00:35
20	264	76	01;15
30	243	89	02:06
40	221	113	02:54
50	208	145	03:31
100	143	254	06:12
150	98	287	08:14
200	52	315	10:20

Tabela 5.6: Caso de teste 3 e influência das iterações externas no tempo e número de *contigs* obtidos pela montagem

Número de Iterações Internas (MAX_EXTERNAS)	Caso de Teste 3		
	Número Contigs	Maior Contig	Tempo (min:seg)
10	374	65	00:47
20	352	71	01;28
30	339	80	02:14
40	328	96	03:05
50	311	107	03:46
100	213	195	06:43
150	157	238	11:20
200	105	296	10:20

Assim como os resultados apresentados na Tabela 5.1 (caso de teste 1) o aumento no número de iterações internas do algoritmo resulta em um menor número de *contigs*. Nas Tabela 5.3 e 5.4 o valor de MAX_EXTERNAS está fixo em 300 e o número de iterações internas está sendo variado. Nas Tabelas 5.5 e5.6 o valor de MAX_INTERNAS está fixo em 100.000 e o número de iterações externas está sendo variado. As Figuras 5.5 e 5.6 mostram o graficamente o comportamento dos casos de teste 2 e 3.

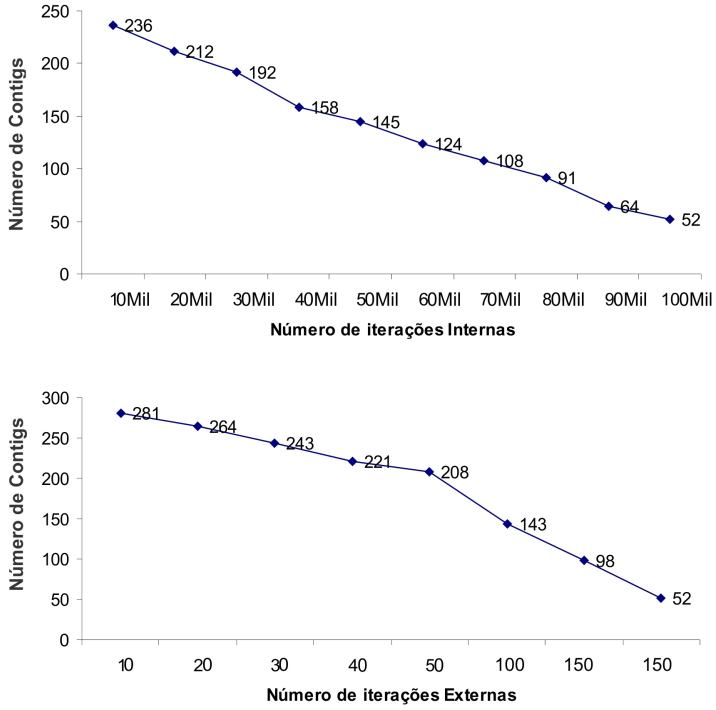


Figura 5.5: Influência no número de iterações internas e externas sobre a montagem do caso de teste 2 de *reads*.

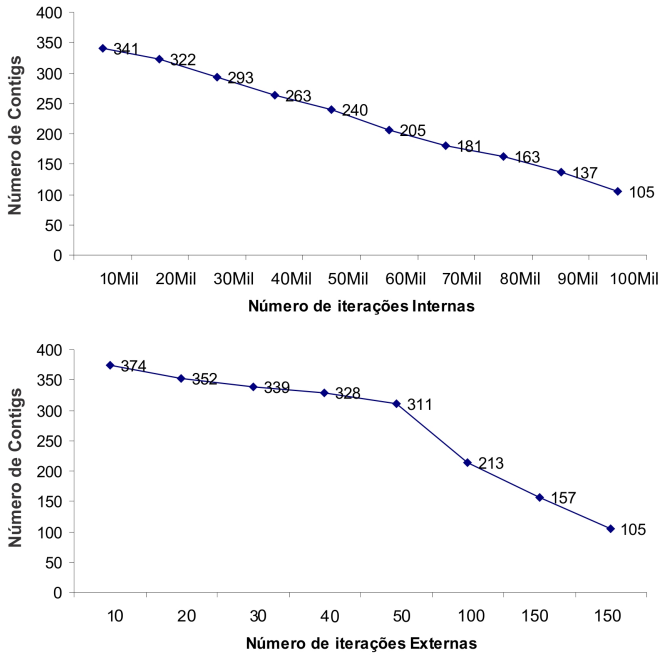


Figura 5.6: Influência no número de iterações internas e externas sobre a montagem do caso de teste 3 de *reads*.

Fica evidente a partir dos testes realizados e interpretação do modelo de montagem proposto, que quanto maior o número de iterações internas e externas, menor é o número de *contigs* ao final da montagem. Entretanto a definição de tais valores só pode acontecer de maneira experimental, visto que o processo sofre influência do tamanho da referência bem como da quantidade e cobertura dos *reads* a serem montados, informações esta que são particulares a cada conjunto a ser montado.

Capítulo 6

Conclusões e Trabalhos Futuros

Os recursos, bem como o tempo necessário para se sequenciar um organismo vem diminuindo a cada dia. As tecnologias atuais de sequenciamento produzem um grande volume de informações que necessitam, por meio de ferramentas computacionais, serem armazenadas e processadas para que o conhecimento seja produzido.

A montagem *de novo* de *reads* é um problema que demanda recursos computacionais devido a sua natureza combinatória. A importância da etapa de montagem de *reads* no contexto de um projeto genoma pode ser confirmada pela variedade de ferramentas existentes com este propósito. A solução ótima para este problema é desconhecida, devido ao seu tamanho do espaço de busca, bem como as característica inerentes ao próprio processo de sequenciamento (produção de *reads*).

Neste trabalho de pesquisa um modelo baseado em algoritmo genético foi proposto e implementado. Neste modelo os indivíduos são compostos por pares de *reads*. Um indivíduo será bem avaliado se seus *reads* compartilharem trechos em comum em suas extremidades em suas extremidades, visto que estes trechos representam sobreposições, o que permite que dois *reads* sejam montados em um só, isto é, sejam substituído pelo consenso da sobreposição entre os mesmos. A cada nova geração do AG, pares de *reads* são montados. Para que a montagem aconteça neste modelo, o AG deve ser executado diversas vezes para que a montagem possa convergir para um único *contig*, ou ao menos para que o número de *reads* não montados diminua.

Os testes apresentados no capítulo 5 mostraram que o modelo proposto converge para a solução do problema de montagem uma vez que a quantidade de *contigs* diminui ao longo das gerações, bem como ao longo das diversas execuções do AG. Entretanto o tempo de execução se mostrou inviável. Estratégias como o pré-processamento do conjunto inicial de *reads* a serem montados, a fim de remover *reads* que tenham o seu conteúdo repetido, a clusterização prévia de *reads* para que *reads* menores contidos em *reads* maiores sejam eliminados, a implementação paralela utilizando a tecnologia CUDA [45] são algumas das estratégias a serem investigadas a fim de viabilizar a utilização da metodologia baseada em AG proposta neste trabalho para a solução do problema de montagem de *reads*.

Referências Bibliográficas

[1] J. Shendure and H. Ji, "Next-generation dna sequencing," 2008.

[2] F. Sanger, S. Nicklen, and A. Coulson, "DNA sequencing with chain-terminating inhibitors," *Proceedings of The National Academy of Sciences of The United States Of America*, vol. 74, no. 12, pp. 5463–5467, 1977.

[3] "Illumina." Disponível em: http://www.illumina.com, acessado em março/2014.

[4] M. Ronaghi, M. Uhlén, and P. Nyrén, "A Sequencing Method Based on Real-Time Pyrophosphate," *Science*, vol. 281, pp. 363–365, July 1998.

[5] M. O'Donnell, *Biologia Molecular: Princípios e Técnicas*. Artmed Editora, 2012.

[6] J. Simpson, "Abyss: A parallel assembler for short read sequence data.," *Genome Research*, 2009.

[7] D.R. Zerbino and E. Birney., "Velvet: algorithms for de novo short read assembly using de bruijn graphs.," *Genome Research*, vol. 18, pp. 821–829, 2008.

[8] R. Li, H. Zhu, J. Ruan, W. Qian, X. Fang, Z. Shi, Y. Li, S. Li, G. Shan, K. Kristiansen, S. Li, H. Yang, J. Wang, and J. Wang, "De novo assembly of human genomes with massively parallel short read sequencing.," *Genome research*, vol. 20, 2010.

[9] S. Boisvert, F. Laviolette, and J. Corbeil, "Ray: Simultaneous Assembly of Reads from a Mix of High-Throughput Sequencing Technologies," *Journal of Computational Biology*, pp. 1519–1533, November 2010.

[10] F. O. S. Hernandez D, François P, "De novo bacterial genome sequencing: millions of very short reads assembled on a desktop computer.," *Genome Research*, vol. 18, pp. 802–809, 2008.

[11] R. Chikhi and G. Rizk, "Space-efficient and exact de bruijn graph representation based on a bloom filter.," in *WABI*, vol. 7534 of *Lecture Notes in Computer Science*, pp. 236–248, Springer, 2012.

[12] B. Chevreux, "Using the miraest assembler for reliable and automated mrna transcript assembly and snp detection in sequenced ests,"

[13] Jeck W. R., Reinhardt J. A., Baltrus D. A., Hickenbotham M. T., Mardis V. M. E. R., Dangl J. L., and Jones C. D., "Extending assembly of short dna sequences to handle error," *Bioinformatics*, vol. 23, 2007.

[14] N. Mullikin JC, "The phusion assembler.," *Genome research*, 2003.

[15] M. J. Chaisson, D. Brinza, and P. A. Pevzner, "De novo fragment assembly with short mate-paired reads: Does the read length matter?," 2009.

[16] J. T. Simpson and R. Durbin, "Efficient de novo assembly of large genomes using compressed data structures," *Genome Research*, vol. 22, pp. gr.126953.111–556, Dec. 2011.

[17] B. Schmidt, R. Sinha, B. Beresford-Smith, and S. J. Puglisi, "A fast hybrid short read fragment assembly algorithm.," *Bioinformatics*, vol. 25, no. 17, pp. 2279–2280, 2009.

[18] A. Dayarian, T. P. Michael, and A. M. Sengupta, "SOPRA: Scaffolding algorithm for paired reads via statistical optimization.," *BMC bioinformatics*, vol. 11, no. 1, pp. 345+, 2010.

[19] Y. Liu, B. Schmidt, and D. L. Maskell, "Parallelized short read assembly of large genomes using de bruijn graphs.," *BMC Bioinformatics*, vol. 12, p. 354, 2011.

[20] G. Luque and E. Alba, "Metaheuristics for the dna fragment assembly problem," *International Journal of Computational Intelligence Research*, vol. 1, pp. 98 – 108, Maio 2005.

[21] S. Schadt, E.E. and A. Kasarskis, "A window into third-generation sequencing," *Human Molecular Genetics*, 2010.

[22] S-C. Fang, Y. Wang, and J. Zhong, "A genetic algorithm approach to solving dna fragment assembly problem," *Journal of Computational and Theoretical Nanoscience*, vol. 2, pp. 1–7, Maio 2005.

[23] P. Pevzner, *Computational Molecular Biology: An Algorithmic Approach*. A Bradford book, MIT Press, 2000.

[24] J. C. Setubal and J. Meidanis, "Introduction to computational molecular biology," 1997.

[25] P. Compeau, P. Pevzner, and G. Tesler, "How to apply de bruijn graphs to genome assembly," *Nature Biotechnology*, vol. 29, pp. 987–991, Novembro 2011.

[26] J.R. Miller, S. Koren., and G. Sutton, "Assembly algorithms for next-generation sequencing data," *Genomics*, vol. 95, no. 6, pp. 315–327, 2010.

[27] E. W. Myers, "Towards simplifying and accurately formulating fragment assembly," *Journal of Computational Biology*, vol. 2, pp. 275 – 290, 2000.

[28] S. Batzoglou, D. Jaffe, K. Stanley, J. Butler, S. Gnerre, E. Mauceli, J. B. Berger, and E. Lander, "Arachne: a whole-genome shotgun assembler," *Genome Research*, 2002.

[29] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein, *Introduction to Algorithms*. The MIT Press, 3rd ed., 2009.

[30] E. Metwally, "Next-Generation Sequence Assembly: Four Stages of Data Processing and Computational Challenges," *PLoS Comput Biol*, vol. 9, Dec. 2013.

[31] P. Green, "Disponível em: Phrap." http://www.phrap.org/.

[32] Sutton G.G, White O., Adams M.D, and Kervalage A.R., "Tigr assembler: A new tool for assembling large shotgun sequencing," in *Genome Science and Tech*, 1995.

[33] X. Huang and A. Madan, "Cap3: A dna sequence assembly program," *Genome Research*, vol. 9, pp. 868 – 877, 1999.

[34] Warren R. L., Sutton G. G., Jones S. J. M., and Holt R. A., "Assembling millions of short dna sequences using ssake," *Bioinformatics*, vol. 23, 2007.

[35] Dohm J. C., Lottaz C,, Borodina T., and Himmelbauer H., "Sharcgs, a fast and highly accurate short-read assembly algorithm for de novo genomic sequencing," *Genome Research*, vol. 17, pp. 1697–1706, Novembro 2007.

[36] R. Linden, *Algoritmos Genéticos, uma importante ferramenta da inteligência computacional*. Brasport, 2th ed., 2008.

[37] J. H. Holland, *Adaptation in Natural and Artificial Systems*. University of Michigan Press, 1975.

[38] D. Simon, *Evolutionary Optimization Algorithms*. Wiley, 2013.

[39] R. Parson, S. Forrest, and C. Burks, "Genetic algorithms for the dna sequence assembly," in *AAAI*, 1993.

[40] G. Luque and E. Alba, "Parallel gas in bioinformatics: Assembling dna fragments," *SCI*, pp. 135–141, 2011.

[41] G. P. R. A. Welch, V. Burland and P. Roesch, "Extensive mosaic structure revealed by the complete genome sequence of uropathogenic Escherichia coli.," *Proceedings of the National Academy of Sciences of the United States of America*, vol. 99, no. 26, pp. 17020–17024, 2002.

[42] W. Huang, L. Li, Eugene W. Myers, and G. Marth, "Art: a next-generation sequencing read simulator.," *Bioinformatics*, vol. 28, no. 4, pp. 593–594, 2012.

[43] D. Zerbino, *Velvet Manual - version 1.1*, Agosto 2008. Disponivel em www.ebi.ac.uk/ zerbino/velvet/Manual.pdf.

[44] G. Rudolph, "Convergence analysis of canonical genetic algorithms," *IEEE Transactions on Neural Networks*, vol. 5, pp. 96–101, 1994.

[45] NVIDIA Corporation, *NVIDIA CUDA C Programming Guide*, June 2011.