UNIVERSIDADE FEDERAL DO PARÁ

CENTRO DE CIÊNCIAS EXATAS E NATURAIS

PROGRAMA DE PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

RENATO RENISON MOREIRA OLIVEIRA

GAVGA: Um Algoritmo Genético para Montagem de Genomas Virais

Dissertação de Mestrado

Belém

2017

RENATO RENISON MOREIRA OLIVEIRA

GAVGA: Um Algoritmo Genético para Montagem de Genomas Virais

Dissertação de Mestrado apresentada ao Programa de Pós-Graduação em Ciência da Computação. Instituto de Ciências Exatas e Naturais. Universidade Federal do Pará.
Área de Concentração: Sistemas Inteligentes

Orientador(a): Prof Dr. Claudomiro de Souza de Sales Júnior

Coorientador(a): Profª Drª Regiane Silva Kawasaki Francês

Belém

2017

Dados Internacionais de Catalogação-na-Publicação (CIP)
Biblioteca Central da UFPA

O48g Oliveira, Renato Renison Moreira.
GAVGA: Um Algoritmo Genético para Montagem de Genomas Virais / Renato Renison Moreira Oliveira. – 2017
71 f. : il.; 30 cm.

Orientador: Claudomiro de Souza de Sales Júnior
Coorientadora: Regiane Silva Kawasaki Francês
Dissertação (Mestrado em Ciências da Computação) - Universidade Federal do Pará, Instituto de Ciências Exatas e Naturais, Programa de Pós-Graduação em Ciência da Computação, Belém, 2017.

1. Algoritmos genéticos. 2. Infecção - Modelos matemáticos. 3. Otimização combinatória. 4. Computação evolucionária. I. Sales Júnior, Claudomiro de Souza de, *orient*. II. Francês, Regiane Silva Kawasaki, *oth*. III. Título.

CDD: 23. ed. 006.31

RENATO RENISON MOREIRA OLIVEIRA

GAVGA: Um Algoritmo Genético para Montagem de Genomas Virais

Dissertação de Mestrado apresentada ao Programa de Pós-Graduação em Ciência da Computação. Instituto de Ciências Exatas e Naturais. Universidade Federal do Pará.
Área de Concentração: Sistemas Inteligentes

Orientador(a): Prof Dr. Claudomiro de Souza de Sales Júnior

Coorientador(a): Profª Drª Regiane Silva Kawasaki Francês

Defendido em: __ /__ /__

Conceito: _____

BANCA EXAMINADORA

_____
Prof. Dr. Claudomiro de Souza de Sales Júnior
Orientador

_____
Prof.ª Dr.ª Regiane Silva Kawasaki Francês
Coorientadora

_____
Prof. Dr. Ronnie Cley de Oliveira Alves
Professor PPGCC - UFPA

_____
Prof.ª Dr.ª Danielle Costa Carrara Couto
Professora - UFPA

_____
Prof.ª Dr.ª Fabiola Pantoja Oliveira Araújo
Professora - UFPA

À minha famíla.

AGRADECIMENTOS

Aos meus amados pais, José Oliveira (Dedé, "*in memorian"*) e Helena Oliveira, por nunca terem medido esforços para me darem um boa educação e ser uma pessoa de bem, sempre me apoiarem nas decisões ousadas da vida, como viajar para outro país.

À minha irmã, Maria Oliveira, que me deu apoio e ficou do meu lado em momentos difíceis, e que mesmo hoje com sua família feita, continua me trazendo muitas alegrias, minha sobrinha Maria Clara.

Aos meus tios, tias, primos e primas que se mantiveram juntos a mim e à minha família em momentos difíceis.

Aos amigos que fiz no antigo CEFET, que me fizeram ver que havia um mundo todo além de Icoaraci, e cuja amizade se mantém forte até hoje: Gizele Abdon, Derick Rosa, Felipe Freitas e Charles Pantoja.

Aos amigos que de alguma forma presenciaram minhas preocupações com o mestrado: Derick Rosa (de novo), Bárbara Souza, Ellton Sales, André Cunha, Elder Patten-Ferreira, Talvâne Lima, Reginaldo Cordeiro, Manoel Afonso, Alessandra Priscila, Angelo Oliveira, Cleidyr Lima, Marissa Brasil, Michell Cruz, Brunelli Miranda, Tiago Davi, Júnior Silva, Victor Negrão, Anderson Furtado, Adrielly Góes, Nicoli Silva, Raíssa Lorena, Leonardo Formento, Yury Souza, Helder Paraense, Arthur Gusmão, Dandara Andrade, Andréa Lopez, Ivan Ikikame e Anne Luize.

Ao meu orientador, Prof. Dr. Claudormiro de Souza de Sales Júnior, por ter me aceitando como mestrando e ter sido um ótimo orientador, sempre aberto às minhas opiniões e sugestões na pesquisa acadêmica e ter sempre confiado em mim. À minha querida coorientadora, Profª. Drª. Regiane Silva Kawasaki Francês, pela sua ótima coorientação neste trabalho, e por, em momentos de dificuldades durante a realização do mesmo, ter indicado o caminho para encontrar a melhor saída. Aos mestrandos orientados pelo Prof. Claudomiro, Manoel Afonso e em especial ao Reginaldo Cordeiro, por terem acompanhado de perto a realização deste trabalho dando dicas importantíssimas e nunca me deixando ficar longe das tarefas pendentes.

A todos os professores que fizeram parte da minha educação acadêmica e que de certa forma me inspiram a ser um bom pesquisador, em especial aos Professores Nelson Neto e Josivaldo Araújo, pelas ótimas dicas dadas durante o período da pesquisa.

Aos meus colegas de trabalho no Instituto Tecnológico Vale, por me ajudarem a crescer no caminho da bioinformática (sem esquecer dos Happy Hours, *cheers*), Guilherme Oliveira, Nelson Carvalho, Hivana Barbosa, José Bittencourt, Ivan Nancucheo, Santelmo Vasconcelos, Gisele Nunes, Bruno Garcia, Eder Pires, Mariana Costa, Leandro Santos, Vitor Cirilo, Celso Oliveira e Leon Cruz.

A todos que porventura não estão com o nome aqui, mas que me ajudaram de alguma forma nesta caminhada, meus eternos agradecimentos.

“Nunca deixem que digam que não vale a pena acreditar num sonho que se tem”

Renato Russo

RESUMO

A bioinformática vem sendo bastante utilizada para análise e tratamento dos dados gerados pelos chamados sequenciadores de nova geração. Genomas virais representam um grande desafio para a bioinformática pelo fato de apresentarem uma alta taxa de mutação, o que os permite formar quasi-espécies em um mesmo hospedeiro infectado, dificultando bastante a montagem e posterior análise desses genomas virais. Neste trabalho, foi implementado e avaliado o desempenho de um algoritmo genético, chamado GAVGA, que realiza a montagem de genomas virais. O processo de montagem do GAVGA inicialmente realiza a clusterização das sequências (*reads*) que compartilham uma *substring* comum (*seed*), sendo então utilizado um algoritmo genético para verificar se há sobreposição entre as *reads* que compõem um mesmo *cluster*, dada uma porcentagem de similaridade mínima. Os resultados do GAVGA foram comparados aos montadores Newbler, SPAdes e ABySS, assim como também ao montador viral VICUNA. Tais resultados confirmaram que o GAVGA montou corretamente os dados, com 98% de similaridade e 99% de cobertura com os genomas de referência, obtendo dados melhores que dos outros montadores. O GAVGA foi implementado em Python 2.7+ e pode ser baixado no endereço https://sourceforge.net/projects/gavga-assembler/.

Palavras-chave: Bioinformática, Inteligência Artificial, Montagem de genomas, Vírus, Algoritmos genéticos.

ABSTRACT

Bioinformatics has grown considerably since the development of the first sequencing machine, being today intensively used with the next generation DNA sequencers. Viral genomes represent a great challenge to bioinformatics due to its high mutation rate, forming quasispecies in the same infected host, which difficult the assembling and further analysis in these viral genomes. In this paper, we implement and evaluate the performance of a genetic algorithm, named GAVGA, through the quality of a viral genome assembly. The assembly process of GAVGA works by first clustering the reads that share a common substring called seed and, for each cluster, checks if there are overlapping reads with a given similarity percentage using a genetic algorithm. The assembled data are then compared to Newbler, SPAdes and ABySS assemblers, and to a viral assembler such as VICUNA, which results confirm the feasibility of our approach, obtaining 98% of similarity and 99% of coverage from the reference genomes assembled. GAVGA was implemented in python 2.7+ and can be downloaded at https://sourceforge.net/projects/gavga-assembler/.

Keywords: Bioinformatics, Artificial Intelligence, Genome Assembly, Genetic Algorithms, Virus.

LISTA DE ILUSTRAÇÕES

LISTA DE QUADROS

LISTA DE TABELAS

LISTA DE SIGLAS

A	Adenina
AG	Algoritmo Genético
BLAST	*Basic Local Aligment Search Tool*
C	Citosina
CPU	*Central Process Unit*
DNA	*Desoxiribonucleic Acid*
G	Guanina
GAVGA	*Genetic Algorithm for Viral Genome Assembly*
GPU	*Graphics Processing Unity*
HIV	*Human Immunodeficiency Virus*
NCBI	*National Center for Biotechnology Information*
OLC	*Overlap Layout Consensus*
SRA	*Short Reads Archive*
T	Timina

SUMÁRIO

1. INTRODUÇÃO

1.1. CONTEXTO

O crescimento da bioinformática começou com o surgimento dos primeiros sequenciadores e, por consequência, dos primeiros algoritmos de montagem (NAGARAJAN; POP, 2013). Desde então, os genomas de milhares de micro-organismos já foram montados e armazenados em bancos de dados públicos. Montagem de genomas, alinhamento/agrupamento de sequências e predição de genes são alguns de muitos objetivos que podem ser alcançados por meio de técnicas computacionais como mineração de dados, aprendizado de máquina e técnicas de clusterização (GOÉS *et al.*, 2014; PALMER *et al.*, 2010).

Ainda não existe nenhum método capaz de obter a informação genética completa contida na molécula de DNA de um micro-organismo, a partir de um único fragmento. Para resolver esse problema, uma molécula de DNA é quebrada de forma aleatória em fragmentos menores, e tais fragmentos são lidos pelas máquinas sequenciadoras, gerando as chamadas *reads* (leituras), que então são montados a fim de se obter o genoma completo original. Essas *reads* são agrupadas, caso elas compartilhem a mesma sequência de nucleotídeos, em estruturas chamadas *contigs*, que também podem ser agrupadas em estruturas maiores chamadas *scaffolds*. As diferentes formas de se montar tais *reads* constituem um problema conhecido como montagem de fragmentos (NAGARAJAN; POP, 2013).

Existem muitos softwares e algoritmos capazes de montar um conjunto de *reads*, sendo que cada um deles realiza a montagem de formas diferentes. A maioria desses softwares se baseiam em dois métodos bem conhecidos, a técnica OLC (*Overlap – Layout – Consensus*) e o Grafo de Bruijn, que usam grafos Hamiltonianos e Eulerianos, respectivamente (LI *et al.*, 2012).

Em particular, genomas virais ainda apresentam um grande desafio para a Bioinformática, devido ao fato de apresentarem uma alta taxa de mutação no seu DNA, cuja probabilidade de ocorrer uma substituição por nucleotídeo por célula infectada (*s/n/c*) varia entre 10^{-8} e 10^{-4} (SANJUÁN *et al.*, 2010), dificultando o processo de montagem dos genomas e criando uma população de quase-espécies (DOMINGO; SHELDON; PERALES, 2012; HENN *et al.*, 2012; POIRIER; VIGNUZZI, 2017). A montagem de genomas virais tem uma

vital importância pois permite entender como tais vírus funcionam, a partir dos seus genes e da estrutura do seu DNA, permitindo o desenvolvimento de vacinas que ajudam a evitar suas ações em organismos hospedeiros (MODJARRAD; KOFF, 2017)**.** Apesar de todo o avanço em tecnologias de sequenciamento, as infecções virais lideram as causas de morte no mundo inteiro (STANAWAY *et al.*, 2016), então se faz importante a criação de novos métodos voltados para a montagem de genomas virais, permitindo sua análise posterior na busca pela compreensão de seus genomas.

Uma outra técnica que atualmente tem apresentado bons resultados em muitas áreas de aplicação é o Algoritmo Genético (AG), uma técnica de inteligência artificial baseada na teoria da evolução. Considerando que um problema específico pode ter muitas soluções aceitáveis, o AG é capaz de obter em uma única execução, várias dessas soluções aceitáveis ou a solução ótima que melhor resolve ou otimiza tal problema. Dessa forma, neste trabalho será mostrado o GAVGA (*Genetic Algorithm for Viral Genome Assembly*), um algoritmo genético para a realização da montagem de genomas virais. Este algoritmo baseia-se principalmente nos principais conceitos da técnica OLC, juntamente com os conceitos envolvidos em algoritmos genéticos.

1.2 TRABALHOS RELACIONADOS

Por se tratar de uma pesquisa interdisciplinar envolvendo algoritmo genético e montagem de genomas virais, esta seção irá apresentar os trabalhos relacionados ao GAVGA nas duas áreas.

1.2.1. Algoritmos Genéticos

No trabalho de (PARSONS, REBECCA J.; FORREST; BURKS, 1995), é usado um algoritmo genético para realizar a montagem de genomas, onde a representação do cromossomo foi feita usando um conjunto de bits que, ao serem transformados em inteiros, representariam o índice de uma *read* a ser montada. Parsons *et al.* consideraram o *fitness* de seus cromossomos como sendo a soma de todos os comprimentos de sobreposição existentes nas *reads* que compõem um cromossomo.

A representação dos cromossomos usados no trabalho de (FRAGA; SILVA, 2014) tentou diminuir a repetição das informações de *reads* em vários cromossomos, fazendo com

que ao invés de todo cromossomo conter todas as *reads* a serem montadas, os cromossomos passariam a conter apenas dois genes que guardariam os índices de duas *reads* a serem montadas. Dessa forma, o número de cromossomos aumenta bastante, uma vez que devem ser criados bem mais cromossomos para que todas as *reads* sejam representadas no problema. Além disso, o valor do *fitness* desses cromossomos também é o tamanho da sobreposição entre as duas *reads* que os compõem. Algoritmos similares podem ser encontrado em diversos trabalhos (HUGHES; HOUGHTEN, 2014; INDUMATHY; MAHESWARI, 2012; KIKUCHI; CHAKRABORTY, 2006).

O problema encontrado nesses trabalhos é que seus algoritmos foram executados com não mais do que 830 *reads*, com o intuito de montar genomas com cerca de 2.500 pares de bases, não sendo realizado nenhum teste para verificar se seus algoritmos seriam capazes de montar um conjunto maior de *reads*.

Pereira, Costa & Digiampietri, (2014) utilizam uma implementação de algoritmo genético para ordenar genomas parciais, ou seja, genomas que já foram montados, mas não terminados, também se utilizando dos *contigs* gerados a partir de outros métodos de montagem. O diferencial para este trabalho é que o GAVGA irá montar as *reads* obtidas pelos próprios sequenciadores, a fim de se chegar ao genoma inicial completo, ao invés de apenas ordenar os genomas.

1.2.2 Montadores de genomas virais

A montagem de genomas virais é uma tarefa difícil devido à vários fatores, como a alta variabilidade na cobertura de sequenciamento ou a presença de *reads* quiméricas, ou seja, *reads* que apresentam sequências tanto dos vírus quanto do hospedeiro, além da alta taxa de mutação dos seus genomas. Na tentativa de resolver tais problemas, vários montadores de genomas virais foram desenvolvidos.

De acordo com a revisão da literatura realizada, o primeiro montador destinado a genoma viral foi o VICUNA (YANG *et al.*, 2012). O VICUNA consiste de 4 passos principais: (i) *Trimmagem* de *reads*, onde adaptadores utilizados no sequenciamento podem estar ligados a algumas *reads*, e devem então ser removidos. (ii) Construção e clusterização de *contigs*, onde as *reads* são agrupadas caso pertençam ao mesmo *contig*, usando *hash* mínimo (YANG; ZOLA;

ALURU, 2011) para inferir similaridade entre as *reads*. (iii) Validação do *contig*, onde dada uma similaridade mínima, *reads* que não possuírem essa similaridade ao serem alinhadas à sequência do *contig* serão removidas do cluster e possivelmente adicionadas a outro cluster, dando origem a um novo *contig*. (iv) Extensão e junção de *contigs*, onde será verificado se os *contigs* previamente formados possuem similaridade, podendo haver sobreposição entre eles. *Contigs* com sobreposição identificada serão unidos de forma iterativa até que não seja mais possível fazer alguma extensão, resultando então nos *contigs* finais.

O PRICE (RUBY; BELLARE; DERISI, 2013) é uma estratégia de extensão iterativa de *contigs* baseada em *reads* pareadas, onde a partir de pequenas sequências de uma referência (*seed*) realiza a construção e extensão dos *contigs*. Primeiramente as *reads* são alinhadas contra as *seeds*, formando então clusters que são montados de forma a gerar *contigs*. Iterativamente, esses *contigs* são alinhados contra as *reads* de forma a encontrar mais *reads* que permitam a extensão do *contig-seed*. *Contigs* que possuam sobreposição de acordo com requisitos mínimos são unidos de forma a gerar *contigs* maiores. O processo é repetido até nenhum novo *contig* ser formado, gerando então o resultado final.

De forma semelhante ao PRICE, o IVA (HUNT *et al.*, 2015) também faz a montagem de seus *contigs* de forma iterativa, com a diferença de que ao invés de usar uma sequência referência, ele parte da sequência *kmer* mais abundante entre as *reads* a serem montadas. Após os *contigs* serem formados, eles podem ser unidos caso tenham similaridade. Pares de *reads* que sejam mapeadas em *contigs* diferentes também podem indicar que dois *contigs* podem pertencer a um mesmo *scaffold*. Quando não for mais possível realizar nenhuma extensão, a montagem é finalizada.

Além destes montadores que apresentam um algoritmo próprio para montagem das *reads*, foram também publicados *softwares* montadores que utilizam nas suas análises algoritmos já publicados, como o VGA (MANGUL *et al.*, 2014), que utiliza o VICUNA. Também foram publicados pipelines que agrupam um conjunto de ferramentas que realizam o tratamento de qualidade das *reads*, a montagem e a ordenação dos *contigs*, como o VirGA (PARSONS, LANCE R *et al.*, 2015), VirAMP (WAN *et al.*, 2015), V-GAP (NAKAMURA *et al.*, 2016) e VirusTAP (YAMASHITA; SEKIZUKA; KURODA, 2016).

Em seu trabalho, (WEISS, 2010) também mostra que já foram desenvolvidos muitos softwares que permitem a montagem de genomas, como o Phred/Phrap/Consed (GORDON; ABAJIAN; GREEN, 1998), Celera Assembler (MYERS *et al.*, 2000), CAP/PCAP (HUANG *et*

al., 2003), ARACHNE (BATZOGLOU *et al.*, 2002), Genome Analyzer (da empresa Illumina) e o GS De Novo Assembler ou Newbler (da empresa Roche, cujo suporte já foi descontinuado). Além dessas, outras ferramenta para montagem de genomas foram desenvolvida, como o MIRA (BIOPHYSIK; SUHAI, 2007), Velvet (ZERBINO; BIRNEY, 2008), AbySS (SIMPSON *et al.*, 2009) e SPAdes (BANKEVICH *et al.*, 2012).

1.2 MOTIVAÇÃO

Com a utilização das técnicas já mencionadas, OLC e Grafo de Bruijn, alguns problemas e limitações acabam surgindo. Um dos problemas e limitações que mais devem ser considerados é o fato de que para que se possa aplicar a montagem de um genoma pela técnica OLC, um caminho Hamiltoniano deve ser formado, ou seja, um caminho no grafo onde todos os vértices sejam visitados uma única vez. Porém, encontrar um caminho Hamiltoniano é considerado um problema NP-Completo, ou seja, ainda não foi criado nenhum algoritmo que consiga resolver esse problema de forma eficiente. Além disso, a utilização da técnica OLC não é aconselhada quando as informações a serem sequenciadas apresentam sequências muito grandes de informações repetidas, podendo gerar uma montagem com pouca qualidade.

Da mesma forma, para que se possa utilizar o Grafo de Bruijn como método de montagem de genomas, o grafo formado com as informações que se desejam sequenciar deve ter um caminho Euleriano, ou seja, um caminho que visite todas as arestas de um grafo uma única vez. Já foi visto experimentalmente que nem sempre é possível se formar um Caminho Euleriano dado um conjunto de *reads*, logo nem sempre o método pelo Grafo de Bruijn pode ser utilizado (NAGARAJAN; POP, 2013).

Além disso, uma variedade de algoritmos de montagem de genomas virais já foi publicada (HUNT et al. 2015; RUBY et al. 2013; YANG et al. 2012). Tais montadores procuram resolver os problemas oriundos da montagem de genomas virais, porém acabam limitando sua utilização. A maioria desses softwares exige que o conjunto de *reads* a serem montados tenham sido gerados por uma tecnologia de sequenciamento específica, além de exigir que apenas conjuntos de dados com alta cobertura de sequenciamento (quanto mais *reads*, melhor) sejam montados, para poder gerar uma boa montagem e assim contornar os problemas ocasionados

pela formação de *reads* quiméricas e pela alta taxa de mutação do genoma viral.

Essas limitações acabam fazendo com que nem sempre esses métodos sejam os mais apropriados para se realizar a montagem de genomas virais, o que torna importante e cabível a aplicação de uma nova técnica no processo da montagem desses genomas. Além disso, no problema da montagem de genoma, os dados usados como entrada são as *reads* extraídas de um organismo, havendo várias formas de organizar essas *reads* em busca da melhor montagem. O algoritmo genético age então na busca da melhor forma de se organizar tais *reads*.

1.3. JUSTIFICATIVA E CONTRIBUIÇÃO À ÁREA

Algoritmos de montagens já existentes (baseados nas técnicas OLC e De Bruijn) não lidam muito bem ao montarem *reads* que derivem de micro-organismos ou populações que apresentem uma alta taxa de variação, como os vírus. Esses algoritmos tradicionais acabam por não saberem separar o que de fato se trata de uma variação nas sequências genéticas ou o que pode ter sido um erro de sequenciamento, podendo gerar ao final resultados não muito corretos (YANG *et al.*, 2012).

Devido a essas limitações, houve um esforço em desenvolver algoritmos de montagens apropriados para genomas virais, como os descritos na Seção 1.2.2 que procuram focar na alta taxa de mutação do genoma viral e na resolução dos problemas referentes às populações de vírus heterogêneas, ou seja, de grande diversidade na população de um único vírus em um único hospedeiro. O problema de muitos desses montadores de genomas virais já existentes na literatura é que eles são muito difíceis de serem usados e podem apenas ser aplicados em conjuntos de dados com um grande número de *reads single-end* ou pareadas.

A utilização de um algoritmo genético na realização da montagem de genomas virais pode vir a apresentar ótimos resultados, uma vez que AG’s apresentam como uma de suas principais características a capacidade de explorar o espaço de soluções possíveis para um determinado problema, de forma a obter a melhor solução. Além disso, nenhum dos trabalhos correlatos apresentados na seção 1.2.1 tentou montar a mesma quantidade de *reads* usada neste trabalho (cerca de 10 mil *reads*).

A contribuição deste trabalho está na disponibilização de um novo algoritmo de montagem de genomas virais, permitindo que os pesquisadores da área possam ter acesso a resultados que não tenham sido oriundas das técnicas OLC e Grafo de Bruijn. É de extrema

importância para a comunidade verificar que outras técnicas e abordagens podem ser utilizadas (e com sucesso) no âmbito da montagem de genomas, o que incita a busca por melhorias e novas abordagens neste tema. O trabalho desenvolvido resultou na disponibilização do montador de genomas virais GAVGA, que a partir da técnica de algoritmo genético realiza a montagem de dados oriundos de sequenciamento viral. Além disso, este trabalho pode servir como base para futuros trabalhos que busquem pela utilização de algoritmos genéticos em tarefas de montagem de genomas virais ou não-virais.

1.4. OBJETIVOS

1.4.1. Objetivo Geral

Este trabalho tem como objetivo desenvolver um algoritmo genético para resolver o problema da montagem de fragmentos virais.

1.4.2. Objetivos Específicos

- Desenvolver e aprimorar o algoritmo genético para realizar montagem de genomas virais, a partir da clusterização de *reads* que apresentam similaridade entre si, representação por permutação dos genes de um cromossomo como sendo as *reads* a serem montadas e *fitness* como sendo a soma das sobreposições entre as *reads* adjacentes em um cromossomo.
- Procurar outros softwares que façam a montagem de genomas genéricos e de genomas virais;
- Executar o GAVGA em dados simulados e reais;
- Comparar resultados obtidos pelo GAVGA com os resultados obtidos pelos softwares montadores já existentes;
- Identificar eventuais pontos de melhoria do GAVGA.

1.5. METODOLOGIA DE PESQUISA

Inicialmente foi realizada uma revisão da literatura, onde buscou-se informações a respeito da montagem de genomas virais, softwares montadores e quais as técnicas utilizadas por esses softwares, além da pesquisa de outros trabalhos em que algoritmos genéticos foram usados na montagem de genomas.

As ferramentas utilizadas para o desenvolvimento do AG foram:

- IDE Eclipse Luna 4.4.1 (x64);
- Linguagem de programação Python 2.7 (x64).

A fim de se avaliar o algoritmo desenvolvido, foram utilizados dois conjuntos de *reads*, uma originada de um sequenciamento real de um genoma viral (a partir de agora chamadas de *reads* reais), e outra originada da simulação de um sequenciamento de um genoma viral (a partir de agora chamadas de *reads* simuladas), a fim de se ter mais controle sobre o resultado do algoritmo. Utilizar um conjunto de dados simulados nos permite saber se os resultados gerados estão próximos ou não do que sabemos ser a verdadeira resposta, neste caso, o genoma de referência.

Foi realizado inicialmente um pré-processamento nos conjuntos de *reads* reais e simuladas. Por se tratarem de sequências virais, foi necessário remover prováveis *reads* contaminantes (originadas do hospedeiro) e para isso foi utilizada a ferramenta BioBloom (CHU *et al.*, 2014). Além disso, também foram utilizadas as ferramentas FASTX-Toolkit (http://hannonlab.cshl.edu/fastx_toolkit) e PRINSEQ (SCHMIEDER; EDWARDS, 2011) para realizar o tratamento de qualidade das *reads*.

Ao final do pré-processamento, o conjunto de *reads* reais e simuladas restantes foi de 6,891 e 9,185, respectivamente, sendo essas *reads* utilizadas como entrada para o GAVGA, a fim de se obter o genoma montado. A avaliação dos montadores foi feita levando em consideração: total de *contigs* montados, tamanho do maior *contig*, total de bases montadas, similaridade e cobertura dos maiores *contigs* com as respectivas referências.

1.7 ESTRUTURA DO TRABALHO

Além desta Introdução, este documento apresenta mais 5 capítulos que discorrerão sobre os assuntos listados abaixo e que fornecerão um melhor entendimento sobre o trabalho realizado. Os capítulos são:

- **Capítulo 2 – Contextualização e terminologias do trabalho:**

  Neste capítulo serão apresentados os métodos mais utilizados para a montagem de genomas: Grafos de *Sobreposição* e Grafos de *Bruijn*. Também será falado sobre a montagem e anotação de genomas virais e suas dificuldades. Por fim será discorrido sobre o funcionamento de algoritmos genéticos.

- **Capítulo 3 – GAVGA, um algoritmo genético para montagem de genomas virais:**

  Neste capítulo será apresentado o GAVGA, um algoritmo genético para montagem de genomas virais, assim como os detalhes que envolvem sua implementação e as considerações a respeito de sua utilização.

- **Capítulo 4 – Resultados e Discussão:**

Neste capítulo será mostrado como se deu a aplicação do GAVGA para dados reais extraídos de um vírus e dados simulados a partir de um genoma viral, assim como a descrição dos resultados e a discussão do mesmo.

- **Capítulo 5 - Conclusão:**

Neste último capítulo serão apresentadas as considerações finais acerca do trabalho realizado, assim como sugestões para trabalhos futuros.

2. CONTEXTUALIZAÇÃO E TERMINOLOGIAS DO TRABALHO

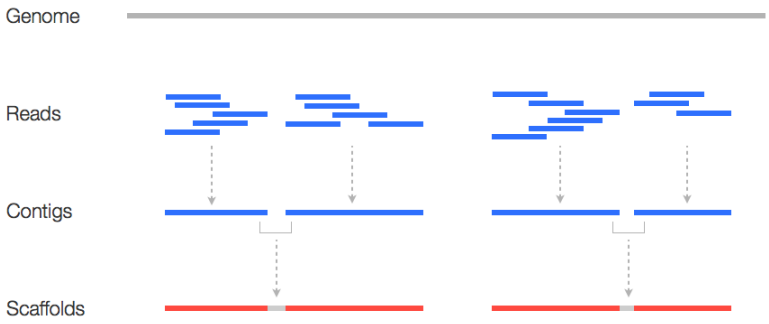
2.1. SEQUENCIAMENTO E MONTAGEM DE GENOMAS

Ainda não foi desenvolvido um sequenciador capaz de obter o genoma completo de um organismo de uma única vez. É por esse motivo que o processo de sequenciamento é realizado. Sequenciar genomas trata-se de obter a informação genética de forma digital a partir de fragmentos do DNA de um organismo, para que depois tais segmentos (já digitalizados) possam ser montados e assim se obter maiores fragmentos do DNA (chamados *contigs*), tornando possível a obtenção do genoma completo.

As duas principais estratégias de sequenciamento existentes na literatura são o sequenciamento dirigido e o randômico (aleatório) (LEMOS; BASÍLIO; CASANOVA, 2003; NAGARAJAN; POP, 2013). A estratégia de sequenciamento que é mais importante para este trabalho é o sequenciamento randômico. Esse sequenciamento, também conhecido como shotgun, submete o DNA a ser sequenciado a um processo que ocasiona a quebra das moléculas em pontos aleatórios (METZKER, 2009) . Ao final do processo de sequenciamento randômico, obtém-se os fragmentos digitalizados de DNA, também chamados de *reads*, que serão de extrema importância para a fase de montagem do genoma.

O sequenciamento dirigido e randômico se diferem na forma em que o sequenciamento é feito, porém, independentemente da estratégia utilizada, nenhuma parte do DNA original deve ser deixada de ser representada pelos fragmentos, ou seja, deve-se conseguir uma boa cobertura (METZKER, 2009; STERKY; LUNDEBERG, 2000). Um exemplo do que seria uma boa cobertura está representado na Figura 1.

Figura 1 - Como *reads*, *contigs* e *scaffolds* estão organizados.



Fonte: Ecology and Evolution Unit Page (2015).

Na Figura 1, a linha que está no topo representa o DNA a ser sequenciado. Já as linhas da segunda camada representam os fragmentos obtidos ou *reads*. Para se obter uma cobertura, também é necessário que haja uma sobreposição suficiente entre os fragmentos. A cobertura do sequenciamento de um genoma pode ser obtida ao se calcular a soma dos pares de base das *reads* geradas e em seguida dividir esse valor pelo tamanho aproximado do genoma. Por exemplo, se ao final de um sequenciamento foi gerado um total de um milhão de *reads*, cuja soma dos pares de base totaliza 20 milhões, e supondo que o genoma do organismo sequenciado seja de ~500,000 bp (pares de base), assim a cobertura do sequenciamento é de 40X.

Após a obtenção dos fragmentos digitalizados pelo sequenciamento, dá-se início ao processo de montagem do genoma. Neste processo, os fragmentos obtidos na fase de sequenciamento são de certa forma agrupados em *contigs* (terceira camada da Figura 1) e estes, por sua vez, se agrupam e formam os *scaffolds* (quarta camada da Figura 2) caso a tecnologia de sequenciamento utilizado tenha gerado *reads* pareadas. A junção dos *scaffolds* é que compõem o genoma completo do organismo, que pode-se dizer que seja o produto final do processo de montagem.

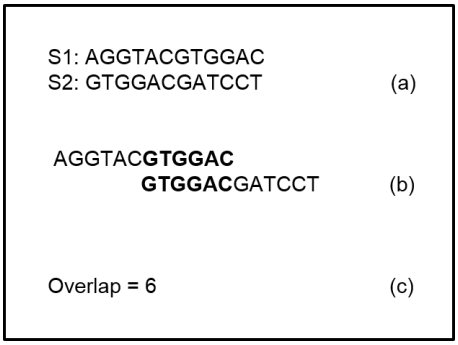
Existem várias técnicas de montagem de fragmentos, dentre elas, duas se destacam mais: OLC (*Overlap-Layout-Consensus*) e Grafo de Bruijn (LI *et al.*, 2012; PEVZNER; TANG; WATERMAN, 2001). A técnica OLC é a que apresenta conceitos mais importantes para um melhor entendimento deste trabalho. No trabalho de (MYERS JR, 2016) é mostrado que a técnica OLC apresenta três passos importantes:

- Detecção de Sobreposição (*Overlap*);
- *Layout* dos Fragmentos;

- Decisão da Sequência Consenso.

Na detecção de sobreposição, tenta-se descobrir qual a sobreposição entre dois fragmentos. Considerando que fragmentos sejam cadeias de caracteres compostas pelas letras (nucleotídeos) “A”, “C”, “G” e “T”, saber a sobreposição entre essas duas cadeias significa saber qual o sufixo da primeira cadeia que é igual ao prefixo da segunda. Na Figura 2 tem-se um exemplo da determinação de sobreposição entre duas sequências (a).

Figura 2 - Exemplo de cálculo de sobreposição entre duas sequências



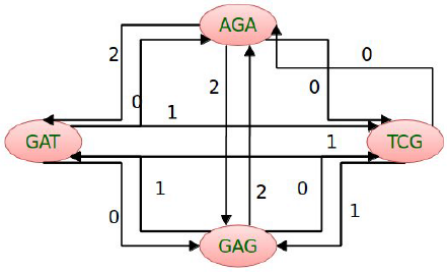
Fonte: Elaborada pelo autor.

Em (b) tem-se a sobreposição entre as duas sequências S1 e S2 (a), e em (c) tem-se o valor em si da sobreposição, que corresponde ao tamanho da sub-cadeia que é sufixo da sequência S1 e prefixo da sequência S2.

(MYERS JR, 2016) afirma que é nessa fase de detecção de sobreposição que se constrói o grafo de sobreposição. Um grafo nada mais é do que um conjunto de vértices ou nós que se conectam entre si através de arestas. Tanto os nós quanto as arestas podem agregar algum tipo de valor, chamado de peso. Além disso, o número total de arestas que tocam em um nó é chamado de grau de um vértice (JURKIEWICZ, 2008).

Um grafo de sobreposição seria então um grafo dirigido onde os nós são representados por fragmentos (sequências) e as arestas que interligam dois nós possuem como peso o valor da sobreposição entre os fragmentos representados em ambos os nós (MYERS JR, 2016). Além disso, trata-se de um grafo completo, ou seja, há arestas entre todos os nós. Na Figura 3 é representado um grafo de sobreposição dadas as sequências AGA, GAT, TCG e GAG.

Figura 3 - Exemplo de um Grafo de Sobreposição.



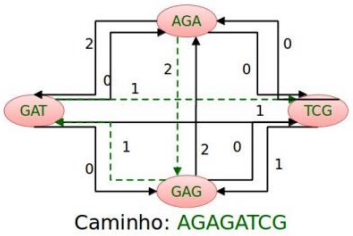
Fonte: Ribeiro (2013).

Na Figura 3, pode-se observar que cada uma das sequências se tornou um nó do grafo, e há aresta entre todos os nós do grafo. O peso das arestas representa o valor da sobreposição entre os nós interligados, sendo o nó da origem da aresta a primeira sequência e o nó destino, a segunda sequência.

Tendo-se construído o grafo de sobreposição, dá-se início ao próximo passo da técnica OLC, a descoberta de caminhos no grafo ou a realização do *Layout* dos fragmentos (MYERS JR, 2016). Nesta fase deve-se descobrir um caminho de Hamilton no grafo de sobreposição que corresponda aos segmentos do genoma. Um caminho de Hamilton é um caminho que visita todos os nós de um grafo uma única vez (GILMORE *et al.*, 1986).

Na Figura 4, tem-se em linhas tracejadas um caminho de Hamilton encontrado no grafo de sobreposição da Figura 3, onde o nó com a sequência “AGA” a origem do caminho.

Figura 4 - Caminho de Hamilton encontrado em um Grafo de Sobreposição.



Fonte: Ribeiro (2013).

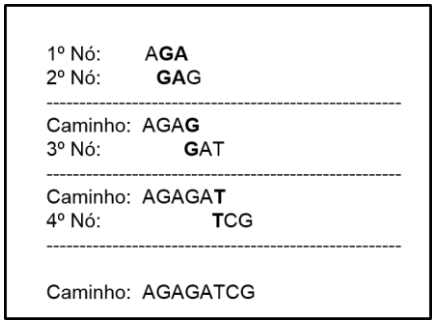
No caso do Grafo de sobreposição, o melhor caminho de Hamilton a ser encontrado será aquele que apresenta a maior soma dos pesos das arestas. Tendo encontrado esse melhor caminho de Hamilton, passa-se para o último passo da técnica OLC, o Consensus. Neste último passo, apenas reconstrói-se a sequência com o caminho de Hamilton escolhido (LI *et al.*, 2012).

Na Figura 4 tem-se a sequência final reconstruída com o caminho escolhido, partido do nó com a sequência “AGA”.

À medida que se vai percorrendo os nós do caminho, realiza-se o procedimento de montagem demonstrado na

Figura 5.

Figura 5 - Processo de Montagem das Sequências do grafo de sobreposição da Figura 6.



Fonte: Elaborada pelo autor.

Figura 5, os caracteres em negrito representam o valor da sobreposição presente nas arestas do grafo da Figura 4 entre as sequências representadas nos nós.

Tanto encontrar um caminho de Hamilton, quanto encontrar o menor caminho de Hamilton são problemas NP-Difícil quando se trata de um problema de otimização, e NP-Completo ao se tratar de um problema de decisão, não existindo ainda algoritmos eficientes que resolvam esse problema (BERTOSSI, 1981; HÉAM; HUGOT; KOUCHNARENKO, 2017).

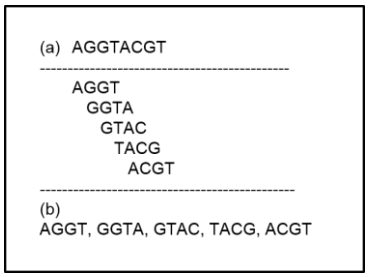
Além disso, caso as sequências que se desejem montar apresentem muitas sequências repetidas, o uso da técnica OLC não será muito eficaz (LI *et al.*, 2012).

Alguns softwares montadores já bem estabelecidos na literatura utilizam conceitos da técnica OLC em seus algoritmos, como é o caso do Newbler, ARACHNE e MIRA.

Diferentemente da técnica OLC, onde os fragmentos ou sequências são os nós do grafo, no grafo de *Bruijn* as sequências que se desejem montar não participam da composição do grafo, mas sim, os espectros de kmers (substring de tamanho *k*) dessas sequências, sendo *k* um valor adotado comumente menor que o tamanho da sequência (PEVZNER; TANG; WATERMAN, 2001). Para um melhor entendimento, a Figura 6 demonstra como se obter o

espectro 4-mers (b) de uma dada sequência (a).

Figura 6 - Espectro 4-mer de uma sequência.



Fonte: Elaborada pelo autor.

No caso da Figura 6, o valor de *k* é 4, retornando então cinco 4-mers (b) obtidas da sequência (a).

Dessa forma, no grafo de Bruijn, os espectros kmer de todas as sequências que se deseja realizar a montagem serão as arestas do grafo, e os nós de cada ponta da aresta irão corresponder a (*k*-1)-mers, onde a direção da aresta irá de um prefixo de tamanho *k*-1 para um sufixo de tamanho *k*-1. Na Figura 7 demonstram-se os nós e arestas formados a partir da 3-mer "ATG".

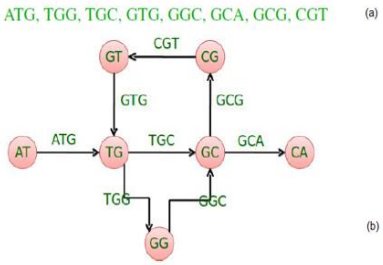
Figura 7 - Exemplo de nós e aresta formados a partir de um 3-mer.

Fonte: Ribeiro (2013).

Nota-se na Figura 7 que o nó de origem da aresta "ATG" é o prefixo "AT" e o nó destino é o sufixo "TG".

Na Figura 8, tem-se então a construção do grafo de Bruijn (b), considerando as 8 3-mers representadas em (a).

Figura 8 - Grafo de Bruijn.

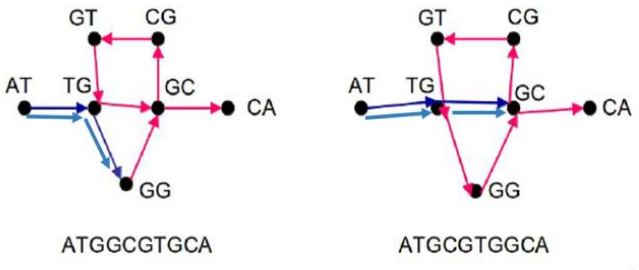


Fonte: Ribeiro (2013).

Após a construção do grafo de *Bruijn*, para que se construa a sequência final, basta apenas encontrar um caminho Euleriano, ou seja, um caminho que percorra todas as arestas uma única vez. Existem algoritmos que resolvem esse problema de forma linear (FLEISCHNER, 1990). Apesar disso, existe a possibilidade de não se achar um caminho Euleriano no grafo, pois para que se encontre um caminho Euleriano, é necessário que haja apenas dois nós semi-balanceados, ou seja, dois nós em que o módulo da diferença entre o número de arestas que entram no nó e o número de arestas que saem do nó é igual a 1 (MILLER et al. 2010). Caso essa condição não seja satisfeita, não se torna possível encontrar um caminho Euleriano no grafo.

Na Figura 9 tem-se o grafo de *Bruijn* com dois possíveis caminhos Eulerianos a serem seguidos.

Figura 9 - Caminhos Eulerianos encontrados no grafo da Figura 8.



Fonte: Ribeiro (2013).

Obtém-se então a sequência final montada ao se percorrer o caminho definido pelo caminho Euleriano. No caso da Figura 9, foram encontrados dois caminhos Eulerianos, que ao serem percorridos, representariam montagens diferentes. As setas duplas de coloração azulada nos grafos da Figura 9 representam os dois diferentes caminhos que podem ser tomados para

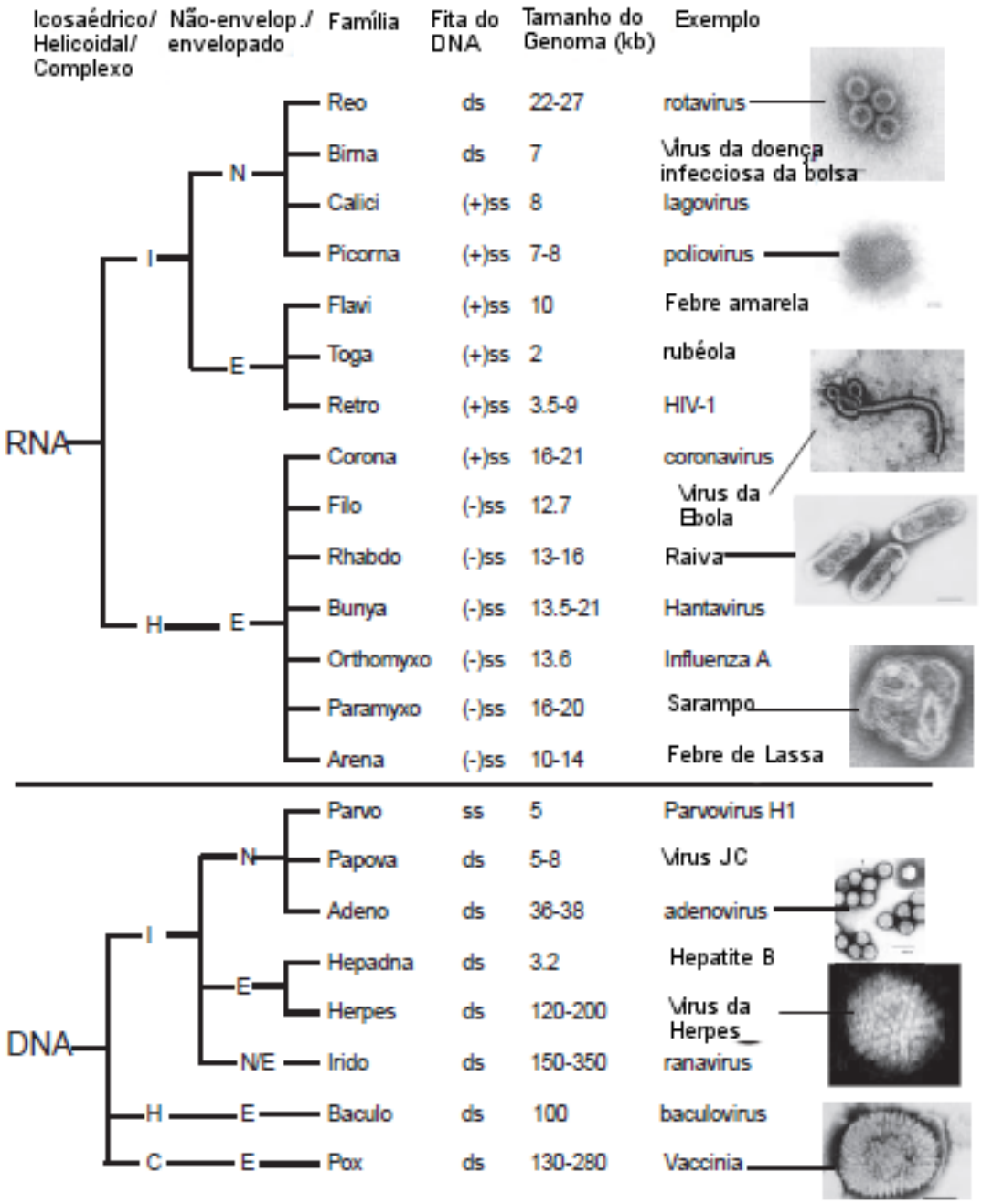
percorrer o grafo.

Dentre os principais softwares que utilizam grafo de *Bruijn*, temos o SPAdes, ABySS e Velvet.

2.2. MONTAGEM E ANOTAÇÃO DE GENOMAS VIRAIS

Vírus são definidos como pequenos parasitas infecciosos que dependem de suas células hospedeiras para poderem se replicar. Genomas virais podem consistir tanto de DNA ou RNA, além disso, essas moléculas podem ser de fita simples (ss), fita dupla (ds), parcialmente de fita dupla (+/-ss), com genomas circulares, lineares ou segmentados (PEVSNER, 2015). A Figura 10 apresenta uma rápida classificação dos tipos de vírus existentes, levando em consideração as características apresentadas anteriormente.

Figura 10 - Classificação dos vírus



Fonte: Adaptado de Pevsner (2015).

Apesar de terem a habilidade de se replicarem e evoluírem, os vírus estão na beira da definição de vida, pois não possuem processos bioquímicos necessários para existirem de forma independente. O vírus de maior genoma conhecido tem cerca de 1 milhão de bases (Mimivirus), enquanto o de menor genoma conhecido tem cerca de 2 mil pares de bases (vírus da rubéola),

como visto na Figura 10 (PEVSNER, 2015).

Os vírus são as entidades biológicas mais abundantes no planeta, apesar de se conhecerem apenas poucos milhares de espécies de vírus (EDWARDS; ROHWER, 2005; PAEZ-ESPINO *et al.*, 2016). Os vírus são capazes de infectar qualquer forma de vida, incluindo bactérias, arqueas e eucariotos, desde fungos até humanos (PRANGISHVILI; GARRETT; KOONIN, 2006).

O ICTVdb é o banco de dados do Comitê Internacional de Taxonomia Viral, que pode ser consultado pela internet no site https://talk.ictvonline.org e que busca regularizar as classificações taxonômicas de vírus. Segundo o ICTVdb, até o ano de 2016 os vírus podiam ser classificados taxonomicamente em 11 ordens, 122 famílias, 35 subfamílias, 736 gêneros e 4404 espécies. A taxonomia tem como premissa representar a filogenia das espécies. No caso dos vírus isso se dificulta, uma vez que por seus genomas possuírem uma alta taxa de mutação, análises filogenéticas e estudos da evolução viral acabam dificultadas (PEVSNER, 2015). Essa alta taxa de mutação acaba fazendo com que em um único hospedeiro, uma única espécie de vírus se replique de forma a constituir vírus com sequências genéticas consideravelmente diferente do genoma do vírus ancestral, formando o que é chamado de *quasi*-espécie (NEUMANN *et al.*, 1998; DOMINGO; SHELDON; PERALES, 2012; HENN *et al.*, 2012; POIRIER; VIGNUZZI, 2017). As ferramentas que são capazes de realizar a montagem de genomas virais estão descritas na seção 1.2.2.

Antes de se realizar a montagem dos genomas, é importante realizar algumas etapas de pré-processamento como a remoção dos contaminantes e o tratamento de qualidade. A remoção dos contaminantes é uma etapa essencial, uma vez que ao se sequenciar genomas virais, é provável que uma parte do genoma do hospedeiro também seja sequenciada, podendo gerar assim um viés caso se deseje apenas obter o genoma viral. Por isso então, é realizado o processo de *screening*, onde a partir de uma base de dados com o genoma do hospedeiro, todas as *reads* sequenciadas que tiverem alta similaridade com o banco de genomas de hospedeiros serão removidas. Esta etapa de remoção de contaminantes pode ser realizada com a ferramenta BioBloom (CHU *et al.*, 2014).

O tratamento de qualidade é importante para a verificação da qualidade das leituras que foram sequenciadas a partir de um genoma. O valor de qualidade é definido pela métrica PHRED (GUTHERY; SALISBURY; PUNGLIYA, 2007), que indica a probabilidade de uma base ter sido sequenciada corretamente e varia de 1 a 64. Valores de PHRED 10, 20 e 30 indicam

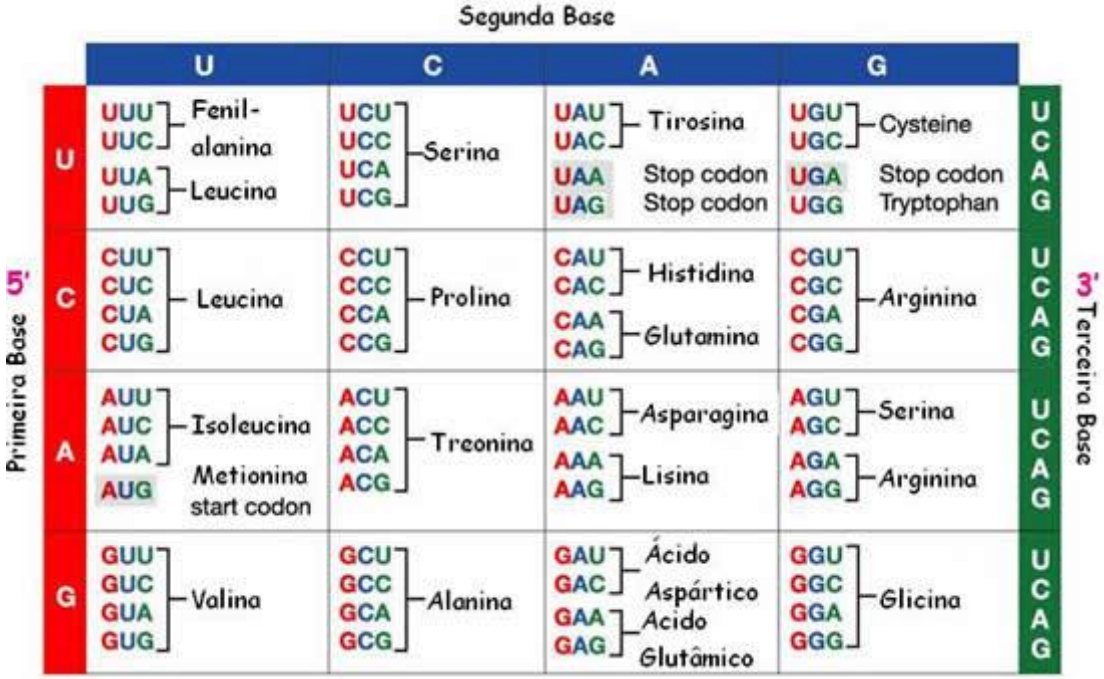
que a base tem 90%, 99% e 99,9% de chance de estar correta, respectivamente. Após o sequenciamento, pode ser que algumas bases das sequências geradas apresentem uma qualidade ruim, precisando ser então podadas, ou seja, cortadas em suas extremidades, deixando apenas bases com qualidade boas. A poda pode ser feita com as ferramentas PRINSEQ (SCHMIEDER; EDWARDS, 2011) e FASTX-Toolkit, onde dado um valor mínimo de PHRED, as bases da extremidade que tiverem uma qualidade abaixo desse valor serão podadas.

Após a poda, ainda assim a sequência pode ainda conter bases de qualidade ruins nas suas regiões mais centrais. Por esse motivo, também se realiza o processo de filtragem, onde um valor de PHRED é determinado para que se a média de valores PHRED de todas as bases de uma sequência for menor que o valor limiar, a sequência toda é descartada. Para a etapa de filtragem, tanto o PRINSEQ quanto o FASTX-Toolkit podem ser usados.

2.2.1. Anotação de Genomas Virais

Genes são as sequências de DNA (ou de aminoácidos) que são responsáveis pela síntese de proteínas no micro-organismo. Proteínas são estruturas compostas de aminoácidos e que desempenham alguma função que pode ou não ser conhecida. Os aminoácidos que compõem as proteínas são formados a partir da combinação de 3 nucleotídeos (códons). Na Figura 11 pode ser visto o chamado código genético, onde está descrito as possíveis combinações de nucleotídeos e quais aminoácidos são formados a partir desses códons (CRICK, 1968).

Figura 11 - Código genético padrão.



Fonte: http://www.sobiologia.com.br.

Nem toda a extensão de um genoma é capaz de codificar uma proteína. Existe 1 códon específico que indica o começo de uma região codificante (códon de iniciação ou *start codon*) e 3 códons que indicam o fim de uma região codificante (códon de parada ou *stop codon*). A região existente entre um códon de iniciação e um códon de parada é chamada de ORF (*Open Reading Frame*) (BADGER; OLSEN, 1999);

Um dos principais objetivos após a montagem de um genoma ser realizado é verificar os genes presentes neste genoma. A este processo é denominado o termo “Anotação”, onde os genes que compõem um genoma são preditos e anotados (RICHARDSON; WATSON, 2012). A predição de genes é o passo inicial, onde as ORFs de um genoma são identificadas e extraídas. Após essa extração, cada ORF é comparada a um banco de dados de genes já conhecidos e quando uma similaridade é encontrada entre uma ORF recém-extraída de um genoma e um gene já conhecido presente no banco de dados, então assume-se que a ORF se trata do gene em questão, sendo possível então conhecer a sua função.

Existem várias ferramentas capazes de realizar a anotação de genomas virais. O GATU (TCHEREPANOV; EHLERS, 2006) é uma ferramenta web que permite realizar a transferência da anotação de genes de uma espécie de vírus já conhecida a uma espécie sem genes anotados,

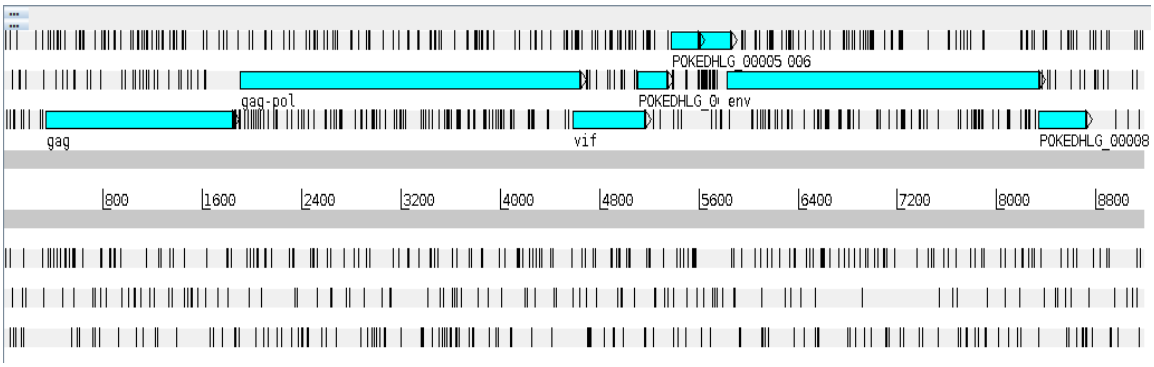
desde que as mesmas sejam bem próximas. Também fornece o conjunto de ORFs presentes no genoma viral alvo que não se encontram no genoma viral de referência, permitindo ao usuário escolher se a ORF é significativa ou não, sendo esse um dos maiores problemas do GATU, pois se torna mais difícil detectar genes novos, já que é necessário haver uma interferência do usuário.

O VIGOR (WANG; SUNDARAM; SPIRO, 2010) também é uma aplicação web desenvolvida para a predição de genes em genomas virais dos subtipos influenza, rotavirus, coronavirus e rhinovirus. A detecção de regiões codificantes é feita por meio da similaridade entre as ORFs com genes já conhecidos de vírus presentes em sua base de dados. Apesar das altas taxas de precisão e acerto que a plataforma VIGOR apresenta (> 99%), ela limita a anotação de genomas virais para apenas vírus que sejam dos subtipos supracitados.

O PROKKA (SEEMANN, 2014) é uma ferramenta *standalone* gratuita destinada à anotação de genomas procariotos, mas que também permite a anotação de genomas virais. O PROKKA usa uma série de ferramentas de bioinformática que isoladamente foram desenvolvidas para a identificação de componentes específicos, mas que ao serem utilizadas em conjunto, geram uma anotação completa do genoma, sem limitar o subtipo do vírus que poderá ser anotado.

Na Figura 12, por exemplo, podem ser vistos os genes presentes em um genoma de HIV (número de acesso NC_001802.1) anotados pela ferramenta PROKKA.

Figura 12 - Genes anotados pela ferramenta PROKKA visualizado na ferramenta Artemis (CARVER *et al.*, 2011).



Fonte: Elaborada pelo autor, por meio da ferramenta Artemis.

2.4. ALGORITMOS GENÉTICOS

Algoritmos Genéticos são técnicas estocásticas de busca e otimização baseada nas teorias de seleção natural e genética de Darwin (NEGNEVITSKY, 2005).

Segundo Darwin, com o passar das gerações de uma população, os indivíduos que possuem uma melhor adaptação ao ambiente têm uma maior probabilidade de perpetuar seu código genético através da reprodução com outros indivíduos. Além disso, essa melhor adaptação pode ser melhorada (ou piorada) caso este indivíduo sofra algum tipo de mutação genética. E dessa forma, através do princípio de seleção, os indivíduos que mais se adaptam a um ambiente vão sobrevivendo, fazendo com que a população se torne uma população mais adaptada ao ambiente.

Em Algoritmos Genéticos, os conceitos usados por Darwin precisam ser representados computacionalmente. Dessa forma, um indivíduo seria então representado por um cromossomo, que iria conter a codificação (genótipo) da possível solução (fenótipo) de um problema que se tenta otimizar ou resolver. Os cromossomos são geralmente implementados na forma de vetores ou lista, onde cada posição do vetor ou da lista são chamadas de genes, e os possíveis valores de um gene são chamados de alelos (ZUBEN, 2000; RIBEIRO *et al.*, 2008).

Zuben (2000) e Pacheco (1999) estabelecem as seguintes características, que devem ser bem definidas para que um Algoritmo Genético seja bem desenvolvido e não apresente erros durante sua aplicação:

- Problema a ser resolvido;
- Representação genética ou genótipo (Codificação de indivíduos);
- Decodificação do indivíduo ou fenótipo;
- Criação da população inicial;
- Avaliação dos indivíduos;
- Seleção dos melhores indivíduos;
- Aplicação de operadores genéticos (cruzamento e mutação);
- Definição de valores dos parâmetros usados no Algoritmo Genético.

Algoritmos Genéticos têm sido aplicados a uma série de problemas de otimização: Otimização de Funções Matemáticas, Otimização Combinatória, Problema do Caixeiro Viajante, Problemas de Otimização de Rotas de Veículos, Otimização em Negócios e Síntese

de Circuitos Eletrônicos (MONTAZERI-GH; POURSAMAD; GHALICHI, 2006; NEUMANN, FRANK; WITT, 2013; SENTHILKUMAR *et al.*, 2014; BAYLIS *et al.*, 2016; ALI *et al.*, 2017)

Pode-se dizer que são inúmeras as aplicações de Algoritmo Genéticos na resolução de problemas, pois segundo Pacheco (1999):

> "AGs são particularmente aplicados em problemas complexos de otimização: problemas com diversos parâmetros ou características que precisam ser combinadas em busca da melhor solução; problemas com muitas restrições ou condições que não podem ser representadas matematicamente; e problemas com grandes espaços de busca. "

A representação de um cromossomo depende muito do tipo de problema que se deseja resolver. O Quadro 1 a seguir resume as melhores formas de se representar um cromossomo, considerando os problemas mais frequentes a serem resolvidos.

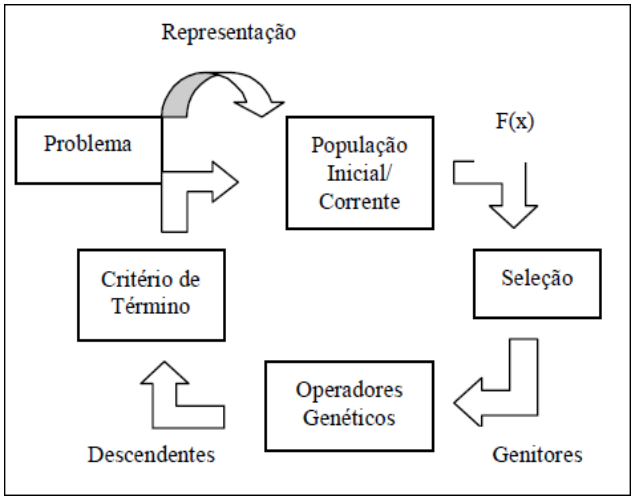
Quadro 1 - Representação de um cromossomo e o tipo de problema a se resolver.

Representação	**Problemas**
Binária	Numéricos, inteiros
Números reais	Numéricos
Permutação de Símbolos	Baseados em Ordem
Símbolos Repetidos	Grupamento

Fonte: Pacheco (1999).

Como a maioria dos problemas que se buscam otimizar utilizando Algoritmos Genéticos são problemas numéricos, as formas de representação mais comumente utilizadas são as representações binária e com números reais (PACHECO, 1999). A Figura 13 resume os principais conceitos descritos até aqui sobre a execução de um algoritmo genético.

Figura 13 - Esquema geral de um algoritmo genético.

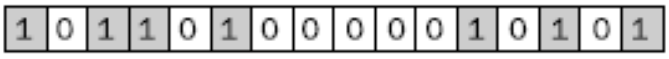


Fonte: Pacheco (1999).

De acordo com a Figura 13, ao se tentar resolver ou otimizar um problema utilizando algoritmos genéticos, cria-se uma população inicial, compostas por cromossomos criados aleatoriamente, dependendo da sua representação. Em seguida, por meio de uma função de avaliação, são selecionados um conjunto de melhores indivíduos, chamados genitores, que passarão pelos operadores genéticos de cruzamento e mutação. Os cromossomos gerados, chamados descendentes, irão compor uma nova população, que passará por todo esse processo novamente, até que um critério de parada seja alcançado. Os cromossomos presentes na última iteração (geração) representam melhores soluções para o dado problema.

A Figura 14 permite de uma forma mais visual verificar como seria a representação binária de um cromossomo em um problema que utilize números inteiros.

Figura 14 - Representação binária de um cromossomo de 16 bits.



Fonte: Negnevitsky (2005).

A utilização da representação binária, porém, apresenta limitações quando os problemas que se tenta otimizar sejam problemas que precisem de parâmetros reais. Neste caso, a representação com números reais ou em ponto flutuante apresentam um melhor desempenho (ZUBEN, 2000).

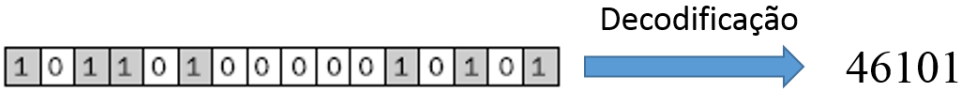
Quando o problema envolve ordenamento de valores, onde geralmente tais valores não

são repetidos, a representação utilizada é a de permutação de símbolos, sendo que estes símbolos podem ser tanto *strings*, quanto números. Neste trabalho, a representação utilizada é a de permutação envolvendo *strings* (PACHECO, 1999).

Segundo Pacheco (1999), decodificar um indivíduo significa construir a solução que um determinado cromossomo representa em seu genótipo para um dado problema. A vantagem de se utilizar a representação binária é que a transformação para um número inteiro ou real acaba sendo mais fácil.

Considerando o cromossomo da Figura 14, por exemplo, a sua transformação para número inteiro seria uma conversão normal de número binário para número inteiro. Essa decodificação é melhor demonstrada na Figura 15.

Figura 15 - Decodificação de um cromossomo.



Fonte: Elaborada pelo autor.

Segundo Zuben (2000), a forma mais comum de inicialização de uma população é a criação aleatória de indivíduos. Supondo que um indivíduo seja representado de forma binária, para cada gene do cromossomo, haveria 50% de chance de o gene ser preenchido com o alelo “1” e 50% de chance de ser preenchido com o alelo “0”.

Zuben (2000) também afirma que se houver um conhecimento inicial a respeito do problema, isso pode ser utilizado na criação inicial do indivíduo. Por exemplo, se for determinado que o número de 1’s e 0’s devem ser iguais, deve-se tomar o cuidado de que todos os indivíduos gerados respeitem essa restrição para que não exista nenhum indivíduo inválido.

Segundo Pacheco (1999), uma boa técnica utilizada nos algoritmos genéticos para encontrar uma boa solução é o preenchimento da população de cada geração (iteração) que for sendo executada com os melhores indivíduos da geração anterior. Dessa forma poderá ser observado que a população acabará convergindo para a melhor solução de um problema.

A avaliação dos indivíduos é feita a partir do cálculo de *fitness* desse individuo, cujo valor é obtido a partir da aplicação de uma função de avaliação sobre o cromossomo analisado. Cada problema possui sua própria função que irá avaliar os seus cromossomos (PACHECO,

1999). Por exemplo, suponha-se que a função de avaliação de um dado problema seja a função f(x) = x^2. Para o cromossomo da Figura 15, cujo fenótipo já sabemos ser 46101, aplicando a função de avaliação, seu fitness seria então 2125302201. Com esse valor de fitness, pode-se saber se este cromossomo é melhor ou pior que outro cromossomo da população.

A seleção é um componente essencial nos algoritmos genéticos. O processo de seleção pode ocorrer em dois momentos distintos durante a execução de um algoritmo genético: na seleção dos pais e na seleção dos sobreviventes.

Na seleção dos pais, são selecionados os melhores indivíduos que irão se reproduzir e gerar novos indivíduos. Essa seleção pode ser feita de várias maneiras, dependendo de como o autor do código deseja, porém pode ser encontrado na literatura cinco principais mecanismos de seleção: seleção por normalização linear, por normalização exponencial, proporcional, com truncamento e por torneios (BLICKLE, 1997). Além disso, o número de pais selecionados para gerar novos filhos pode ser uma porcentagem do tamanho total da população, e não o tamanho da população toda. Adotar esse percentual é típico da abordagem *Steady-State* (LUKE, 2009), onde não é toda a população que é substituída pelos novos cromossomos gerados, mas apenas parte dela, permitindo uma maior diversidade na população de soluções.

Na seleção por torneios, um conjunto de indivíduos é aleatoriamente escolhido de uma população, e o indivíduo com melhor fitness é selecionado. Esse processo é realizado até que o número de indivíduos escolhidos seja o mesmo do tamanho da população. Ao fim deste processo, os operadores genéticos de cruzamento e mutação serão aplicados a esses cromossomos selecionados.

O outro tipo de seleção, a dos sobreviventes, é aplicada quando os operadores genéticos já tiverem sido aplicados a um conjunto de cromossomos pais selecionados. Essa seleção irá definir quais indivíduos passarão para a próxima geração. Comumente, há duas formas de se definir a seleção dos sobreviventes. A primeira forma é definindo que os melhores indivíduos, independentemente de serem indivíduos pais ou indivíduos filhos, irão compor a nova população da geração seguinte (abordagem elitista). A segunda forma é definindo que apenas os indivíduos filhos irão compor a nova população. Na abordagem elitista, todos os cromossomos da geração passada e as proles geradas são ordenadas de acordo com seus *fitness*. Se *c* cromossomos filhos foram gerados, então os *c* cromossomos com o pior *fitness* são removidos da população e os restantes passam para a próxima geração.

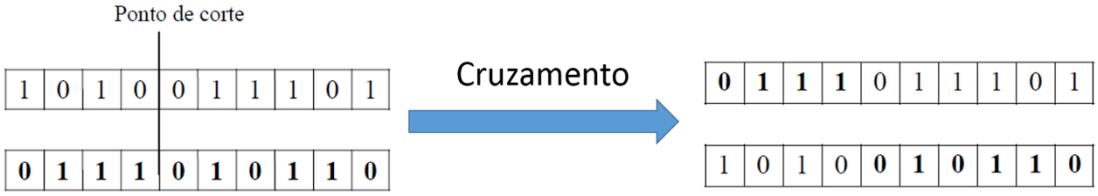
A utilização de operadores genéticos é muito importante para que os indivíduos venham

a ter uma grande variabilidade genética, permitindo assim que o espaço de busca da solução seja percorrido na sua maioria. Zuben (2000) afirma que os operadores genéticos mais comumente utilizados em algoritmos genéticos são o cruzamento, que corresponde a recombinação (cruzamento) de indivíduos, e a mutação.

Para que o cruzamento seja realizado, dois indivíduos, já previamente selecionados na seleção dos pais, são aleatoriamente escolhidos e a partir da troca de informação genética entre esses cromossomos, são gerados dois novos indivíduos contendo informações genéticas de ambos os cromossomos pais. Geralmente também se aplica uma taxa de cruzamento (Pc) comum a todos os indivíduos (ZUBEN, 2000).

Há várias formas de se realizar o cruzamento entre dois indivíduos. A forma mais comum é o cruzamento de um ponto, onde um mesmo ponto é escolhido aleatoriamente para dois indivíduos previamente selecionados, e os genes que ficam depois desse ponto de corte são trocados entre os pais, gerando então dois indivíduos filhos com nova informação genética, pertencentes a ambos os pais. A Figura 16 permite uma visão gráfica de como se dá esse processo de cruzamento.

Figura 16 - Cruzamento de dois indivíduos com um ponto de corte.



Fonte: Elaborada pelo autor.

O operador de mutação é aplicado a todos os cromossomos, causando a modificação de um ou mais genes do mesmo. Para cada gene de um determinado cromossomo, é verificada a probabilidade deste gene sofrer mutação, através da taxa de mutação (*Pm*, geralmente menor que 1%). Para Zuben (2000), a ideia intuitiva por trás da aplicação da mutação aos cromossomos é garantir uma variabilidade extra à população, mas sem que seja destruído por completo as boas soluções já obtidas.

Dado um cromossomo binário, por exemplo, se for determinado que um gene que apresente valor “0” será mutado, após o processo de mutação, este gene passará a ser “1”.

Na execução dos algoritmos genéticos, existem parâmetros que são constantemente

verificados e controlam todo o processo evolucionário, como a probabilidade de cruzamento e a probabilidade de mutação. Além destes parâmetros, Pacheco (1999) também apresenta:

- Tamanho da População: número de cromossomos que irão representar soluções do problema que se tenta otimizar ou resolver.
- Número de Gerações: número de iterações em que o processo de seleção, cruzamento, mutação e avaliação serão realizados, constituindo a evolução do algoritmo genético.

Geralmente o último parâmetro é utilizado como critério de parada para a execução do algoritmo genético.

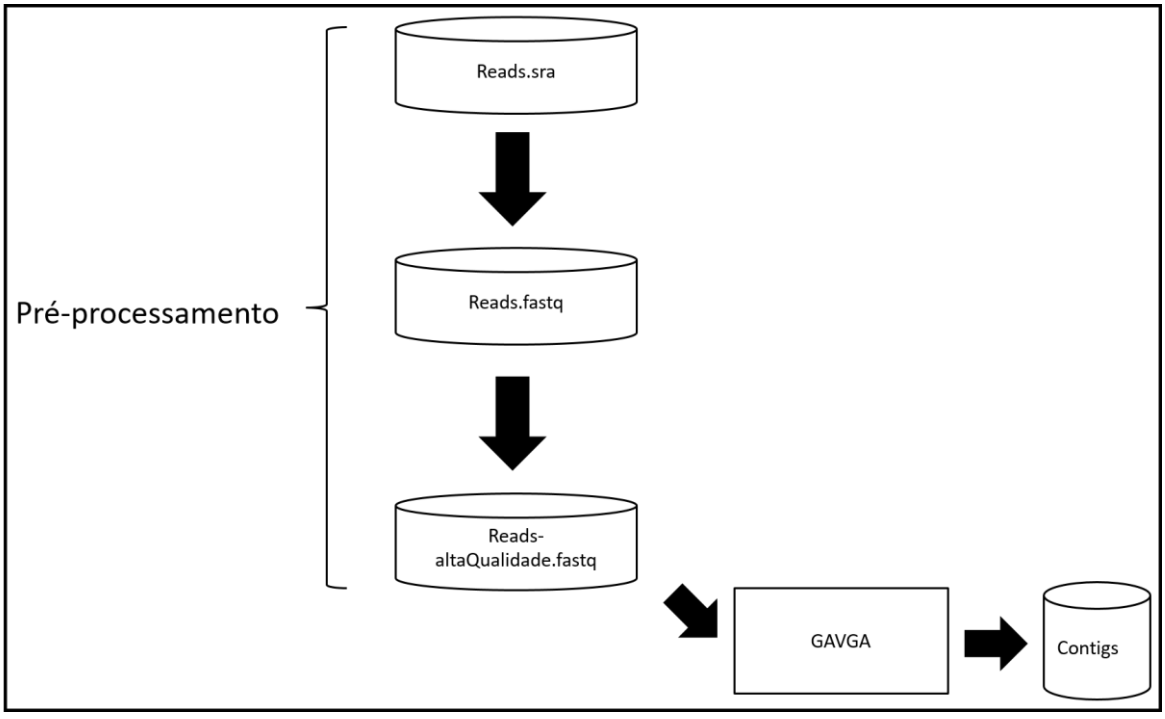
Para os parâmetros de probabilidade de cruzamento e mutação, pode ser também aplicada a estratégia de Rechenberg (1973), onde tais probabilidades aumentam ou diminuem durante a execução do algoritmo, ao ser detectado que o AG se aproximou ou não de uma solução ótima. Por exemplo, se 1/5 da prole tiver um *fitness* melhor do que os pais selecionados em uma geração, isso significa que a solução ótima (local ou global) está longe de ser encontrada e as taxas dos operadores deve ser incrementada, dessa forma acelerando a convergência do algoritmo. Caso contrário, a taxa dos operadores é decrementada, visto que a solução está próxima de ser encontrada.

3. GAVGA: UM ALGORITMO GENÉTICO PARA MONTAGEM DE GENOMAS VIRAIS

Com os motivos apresentados nas seções 1.2 e 1.3, foi desenvolvido um algoritmo genético para montagem de genomas virais, chamado GAVGA (do inglês, *Genetic Algorithm for Viral Genome Assembly*).

Na Figura 17 é dada uma visão geral do algoritmo desenvolvido, desde o pré-processamento realizado nos dados até a aplicação do AG. As subseções seguintes são referentes às etapas apresentadas na Figura 17.

Figura 17 - Visão geral do pré-processamento realizado nos dados e da aplicação do GAVGA.



Fonte: Elaborada pelo autor.

3.1 PRÉ-PROCESSAMENTO DOS DADOS

Para fins de teste, foram baixados do NCBI os dados referentes às *reads* extraídas do *Human Immunodeficiency Virus* 1 (HIV-1, número de acesso ERR846528), totalizando 26,442 *reads*. Também foi feita a simulação de sequenciamento do genoma do vírus da Herpes (Herpesvirus, número de acesso NC_001806.2) utilizando a ferramenta ART (HUANG et al. 2012), gerando um total de 10,724 *reads*, com uma cobertura de 20X. O modelo escolhido para a simulação do sequenciamento foi o modelo de erros do sequenciador 454, gerando *reads* com uma média de 400 bp e com desvio padrão de 20bp.

As *reads* reais foram baixadas do NCBI (número de acesso: ERR846528) no formato SRA (*Short Read Archive*). Assim como os outros montadores de genomas virais, o GAVGA também precisa de um arquivo no formato FASTQ para poder ser executado. Sendo assim, o arquivo SRA baixado foi convertido para o formato FASTQ com o uso da ferramenta SRAToolkit (SHERRY, 2012). Como as *reads* eram pareadas, usou-se a ferramenta PEAR (ZHANG, 2014) para fazer a junção dos pares de *reads* e se obter apenas uma sequência ao invés de um par.

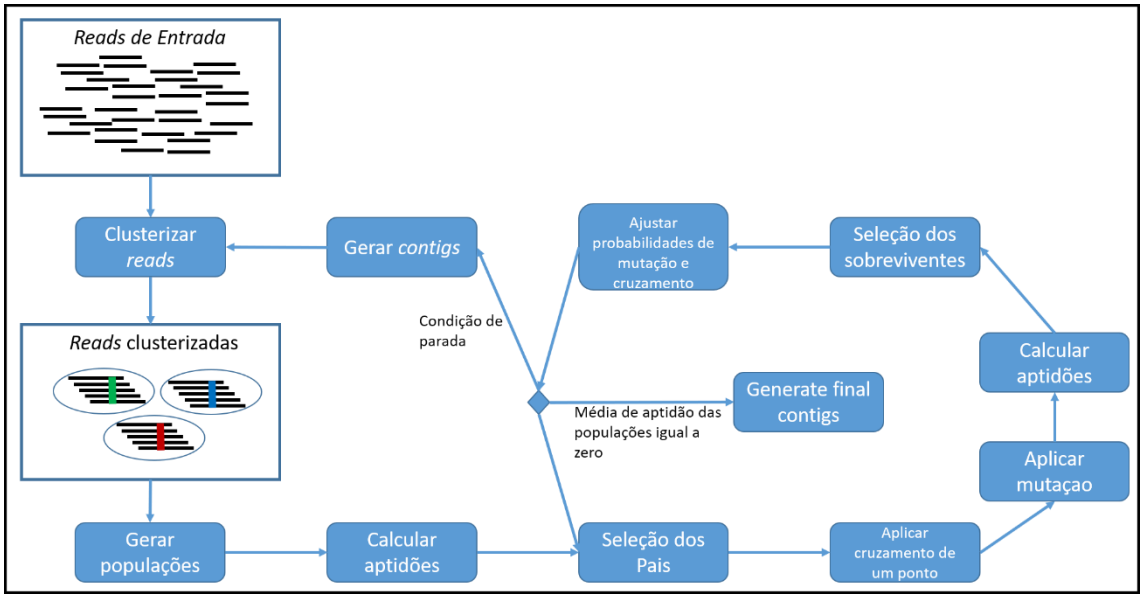
Após o sequenciamento de genomas virais, as *reads* geradas podem apresentar *reads* contaminantes oriundas dos organismos hospedeiros. Tais *reads* foram removidas pela ferramenta BioBloom (CHU, 2014).

A ferramenta PRINSEQ (SCHMIEDER; EDWARDS, 2011) foi utilizada para eliminar bases de qualidade ruins (poda), utilizando uma janela deslizante de 10 pares de base em ambas as extremidades das *reads*, e eliminando tais regiões da janela caso a média de qualidade fosse menor que PHRED 20. Outros tamanhos de janelas foram usados, porém o que gerou melhor resultados foi a janela de tamanho 10, sendo este tamanho usado durante o tratamento de qualidade. Caso a *read* fique com um tamanho menor que 20 pb, a mesma também será descartada. Após a *trimmagem*, foi utilizada a ferramenta FASTX-Toolkit para descartar *reads* que tivesse menos de 80% de suas bases com qualidade acima PHRED 20.

3.2. OPERADORES E COMPONENTES DO GAVGA

O GAVGA foi implementado na linguagem Python e pode ser usado em qualquer sistema operacional que tenha suporte ao Python 2.7+. Na Figura 18 pode se ter uma visão geral dos passos realizados pelo GAVGA que serão descritos com mais detalhes a seguir.

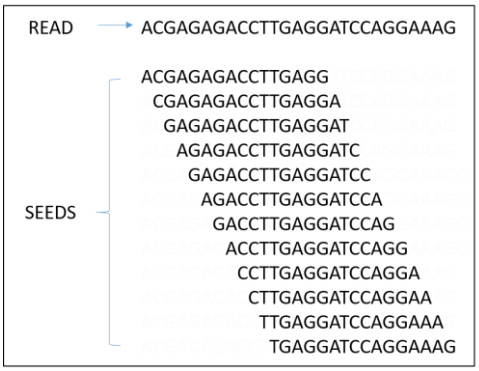
Figura 18 - Diagrama de atividades do algoritmo genético para montagem de genomas virais (GAVGA).



Fonte: Elaborada pelo autor.

Inicialmente, as *reads* são clusterizadas em grupos que compartilham uma mesma *seed* (uma sequência da *read* de tamanho fixo e menor). Neste processo, GAVGA seleciona uma *read* do conjunto de entrada e todas as suas *seeds* são extraídas, ou seja, todas as suas subsequências de um dado tamanho fixo (padrão é 16 bp), com um deslize (*step*) também de tamanho fixo (padrão é 1 bp). Na Figura 19 tem-se um exemplo de como as seeds são obtidas a partir de uma determinada read.

Figura 19 – Exemplo de obtenção das seeds de tamanho 16 a partir de uma dada read.

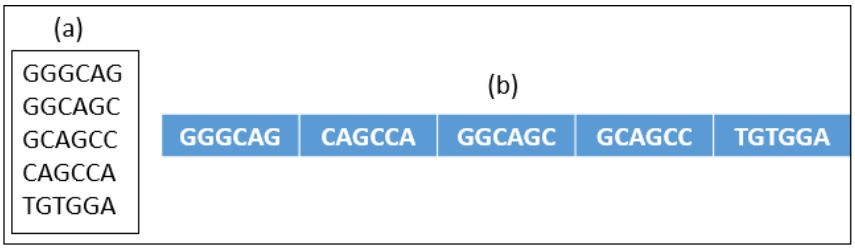


Fonte: Elaborada pelo autor.

Quaisquer *reads* do conjunto de entrada que compartilhar pelo menos uma de suas *seeds* serão agrupadas, formando *clusters*. Se um *cluster* contiver menos de 10 *reads* ou se uma *read* não compartilhar nenhuma *seed*, as mesmas serão reunidas em um *cluster* chamado de "*Reads* restantes".

Cada *cluster* formado irá representar uma população no AG. Os cromossomos que irão compor uma população são representados pela estrutura de dados dicionário onde a chave será a sequência de nucleotídeos de uma *read* e o valor será a posição que essa *read* ocupa no cromossomo. Se o número de *reads* a serem montadas for *n*, então o cromossomo terá *n* posições (genes) e cada uma irá armazenar uma *read* do arquivo de entrada. A Figura 20 mostra como um cromossomo é representado no GAVGA.

Figura 20 - Representação de um cromossomo com *reads* fictícias.



Fonte: Elaborada pelo autor.

Na Figura 20 é mostrado um arquivo simples (***reads* de entrada**) contendo apenas 5 *reads* fictícias (a) e como um cromossomo pode ser representado contendo tais *reads* (b). Para criar uma população de cromossomos (**gerar população**), as *reads* do arquivo

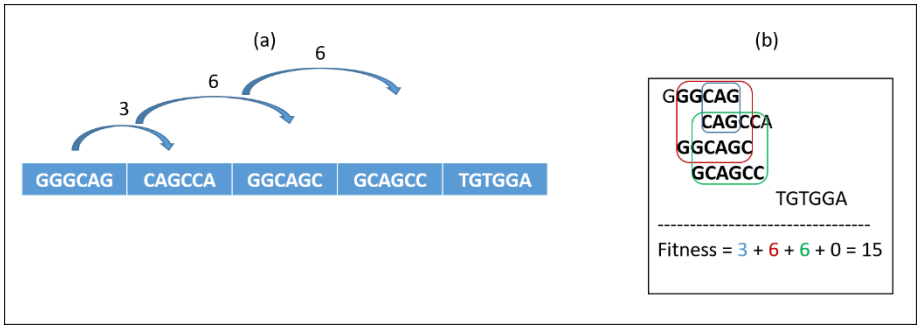
de entrada são aleatoriamente escolhidas sem repetição para compor cada posição do cromossomo.

Para obter a informação representada no cromossomo da Figura 20b, é calculado o seu *fitness* como sendo a soma total das bases que se sobrepõem entre *reads* adjacentes (genes). O algoritmo abaixo demonstra como ocorre seu cálculo:

```
1.  soma_total =0
2.  num_genes=número de genes do cromossomo.
3.  gene0 = cromossomo[0]
4.  i=1
5.  enquanto i < num_genes faça{
6.          gene1 = cromossomo [i]
7.          Se tem_sobreposição(gene0, gene1):
8.              soma_total+=sobreposição(gene0, gene1)
9.              gene0 = monta_contig(gene0, gene1)
10.         Senão:
11.             gene0 = cromossomo[i]
12.         i++
13. fitness = soma_total
```

Seja *i* uma posição do gene de um cromossomo que varie em um intervalo [0, *n*], onde *n* é o total de genes, se houver sobreposição entre a *read* do gene *i* com a *read* do gene *i+1* (linha 7), então o comprimento da sobreposição existente é somado ao valor da soma total de sobreposição (linha 8) e os genes são montados, retornando uma sequência maior (*contig*) que contenha os genes *i* e *i+1* sobrepostos (linha 9). Também é avaliado se existe sobreposição entre o *contig* formado anteriormente com o gene *i+2, e assim por diante, até que* o último gene do cromossomo seja avaliado. Na Figura 21b, por exemplo, o *fitness* do cromossomo representado na Figura 21a é calculado.

Figura 21 - Cálculo do Fitness de um cromossomo.



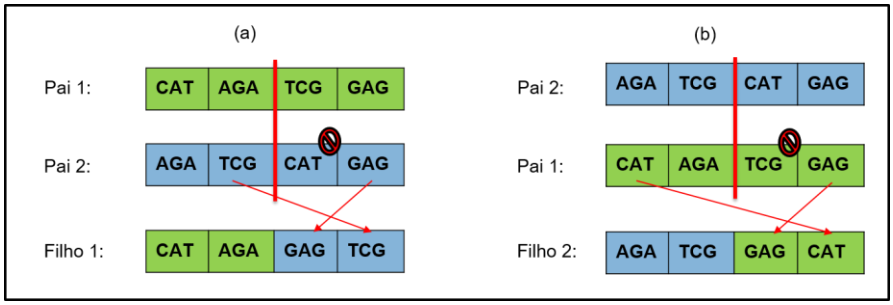
Fonte: Elaborada pelo autor.

No GAVGA, quanto maior o *fitness*, melhor é a sobreposição das *reads*, resultando em uma melhor montagem, dando assim uma maior confiabilidade aos *contigs* resultantes.

No GAVGA, a seleção dos pais é feita por torneio, onde, como descrito antes, *t* (parâmetro ajustável) cromossomos da população são selecionados aleatoriamente e o cromossomo com o melhor *fitness* é selecionado como um futuro pai. Esse processo de seleção é repetido até que se alcance um número de cromossomos igual a 40% do tamanho da população, seguindo a abordagem *Steady-State*.

Após a seleção dos pais, o processo de cruzamento é iniciado, onde os cromossomos selecionados previamente serão escolhidos de forma aleatória com o objetivo de gerarem uma prole. Na Figura 22 é demonstrado o processo de cruzamento entre dois cromossomos no GAVGA.

Figura 22 - Processo de cruzamento entre dois cromossomos.



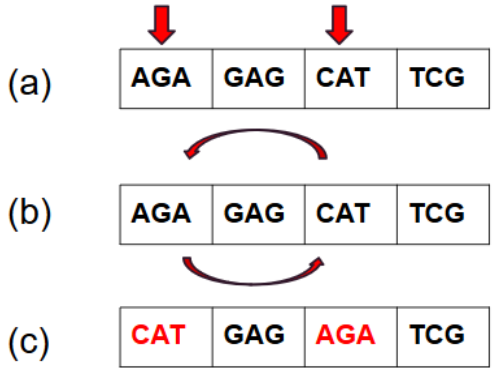
Fonte: Elaborada pelo autor.

O método de cruzamento demonstrado na Figura 22 é o cruzamento de um ponto

(*one-point crossover*) aplicado a problemas de permutação, onde um ponto de corte no cromossomo é escolhido aleatoriamente para ambos os pais, como indicado pelas linhas vermelhas verticais. Genes do cromossomo Pai 1 que estiverem antes desse ponto de corte serão passados para o filho 1, enquanto que os genes restantes do filho 1 serão preenchidos pelos genes do cromossomo Pai 2 sem que haja repetição de *reads*. O segundo filho é gerado de forma similar (Figura 22b).

Em seguida, a prole é submetida ao processo de **mutação**. Na Figura 23 é mostrado como a mutação ocorre no GAVGA.

Figura 23 - Processo de mutação de um cromossomo.



Fonte: Elaborada pelo autor.

Dois genes de um cromossomo são escolhidos aleatoriamente (Figura 23a), permutados (Figura 23b), gerando assim um novo cromossomo (Figura 23c).

No GAVGA, adotou-se a abordagem elitista para a seleção dos sobreviventes. Além disso, adotou-se também que as probabilidades para a taxa de cruzamento estariam entre 0,6 e 0,9 e as probabilidades para a taxa de mutação, entre 0,3 e 0,6. Esses valores foram utilizados como intervalos devido à aplicação da estratégia de Rechenberg, sendo que os maiores valores são os valores iniciais para cada probabilidade, sendo incrementados ou decrementados de acordo com a estratégia de Rechenberg, mas nunca ultrapassando seus limites.

Os parâmetros referentes ao tamanho da população, taxas de cruzamento e mutação, comprimento mínimo de sobreposição, similaridade mínima da sobreposição, número máximo de gerações, ringue do torneio (número de cromossomos a serem selecionados na seleção dos pais) e número de execuções do algoritmo podem ser

ajustados no arquivo de configuração que acompanha o GAVGA.

As condições de parada do GAVGA podem ser atingidas de três formas: (i) se o número máximo de gerações for atingido ou (ii) se a média do *fitness* de uma população não mudar em 30 gerações, é escolhido o melhor cromossomo de cada população, gerados seus *contigs* que então são reclusterizados, dando início a uma chamada “era”. Por outro lado, (iii) se a média do fitness de todas as populações for igual a zero, indicando que não há mais sobreposições a serem feitas entre as sequências, o GAVGA é finalizado gerando assim o resultado final.

O conceito de “era” é análogo ao conceito de geração de um AG. Como no GAVGA há várias populações, e sempre há um momento de reclusterização, toda vez que esse processo de reclusterização é feito, é dado início a uma nova era no GAVGA.

4. RESULTADOS E DISCUSSÃO

Para avaliar os resultados que o GAVGA é capaz de gerar, foram utilizados 2 conjuntos de testes, descritos na seção 3.1. Os resultados do GAVGA para ambos os conjuntos de testes foram comparados aos resultados obtidos pelos montadores Newbler, SPAdes (versão 3.10), AbySS e VICUNA.

Também foi realizada a comparação contra *softwares* montadores específicos para genomas virais. Para que essa comparação fosse justa, foi decidido comparar o GAVGA apenas aos montadores que não oferecessem nenhuma limitação quanto aos requisitos mínimos de execução e que tivessem um algoritmo próprio para montagem, sem utilizar algoritmos de terceiros. Portanto os *pipelines* VirusTAP, VirAMP, V-GAP e VirGA e o montador VGA não foram utilizados como forma de comparação, uma vez que utilizam algoritmos de terceiros para realizar montagem dos genomas. Além disso, o montador IVA não foi utilizado na comparação dos resultados por requisitar dados oriundos de uma tecnologia específica de sequenciamento (Illumina) e não ter gerado resultados ao utilizar os conjuntos de testes como entrada, quando executado . Por fim, o montador PRICE não foi utilizado para fins de comparação por precisar de uma sequência de referência para iniciar a montagem, ficando assim impossibilitado de descobrir novos genomas de vírus.

Das ferramentas voltadas para montagem de genomas virais, apenas o VICUNA satisfez todos os requisitos por utilizar algoritmo de montagem próprio e não ter limitação sobre a tecnologia de sequenciamento utilizada. Todas as ferramentas de montagem foram executadas utilizando apenas um *thread* de processamento com memória compartilhada.

Os parâmetros do AG usados nos testes com dados reais e simulados estão apresentados no Quadro 2 a seguir.

Quadro 2 - Características do AG desenvolvido.

Tamanho da População	Testes feitos com 2, 6 e 10 cromossomos.
Representação	Permutação de *Strings.*
Cruzamento	Cruzamento de um ponto, com Pc = [0.6 - 0.9]
Mutação	Swap, com Pm = [0.3 - 0.6]
Seleção dos Pais	Torneio (Ringue = 5)
Steady-State	Apenas 40% da população pode cruzar
Seleção dos Sobreviventes	Elitista (Remove os piores indivíduos)

4.1 CONJUNTO DE *READS* REAIS

O arquivo FASTQ de *reads* reais baixadas continha 13.221 *reads* pareadas que foram extraídas pelo sequenciador Illumina MiSeq. Após o pré-processamento, restaram apenas 6.891 *reads*, totalizando 1.173.484 bp (1.2 Mbases), com uma cobertura de 130X, assumindo que o genoma de referência do HIV tenha 9.000 bp.

Os montadores Newbler, GAVGA e VICUNA tiveram seus valores de comprimento mínimo de sobreposição e similaridade mínima de sobreposição configurados para 40 bp e 90%, respectivamente. Os montadores SPAdes e ABySS tiveram seus valores de *kmer* configurados para 41, de forma que a sobreposição entre os *kmers* sejam de 40 bp e a comparação entre os softwares seja mais justa.

Por ser um AG, o GAVGA consequentemente se torna um algoritmo estocástico, por isso é recomendado configurar o número de execuções para > 1, para que assim o algoritmo combine os *contigs* obtidos em cada execução, melhorando os resultados finais. Para aumentar a confiança dos resultados do GAVGA, o número de execuções do mesmo foi configurado como sendo 2. Além disso, para fins de dados estatísticos, as duas execuções foram executadas 10 vezes em um computador com processador *Intel core i5, quad-core,* 3.10GHz, com 8GB de memória RAM usando o Sistema Operacional Ubuntu 14.04. Os dados apresentados na Tabela 1 mostram a média e desvios padrão dos resultados obtidos nas 10 execuções do GAVGA com 3 diferentes tamanhos de população (2, 6 e 10 indivíduos). Newbler, SPAdes, AbySS e VICUNA foram executados apenas uma vez, já que seus algoritmos são determinísticos.

Tabela 1 - Resultados estatísticos do GAVGA, Newbler, SPAdes, AbySS e VICUNA ao montar *reads* reais.

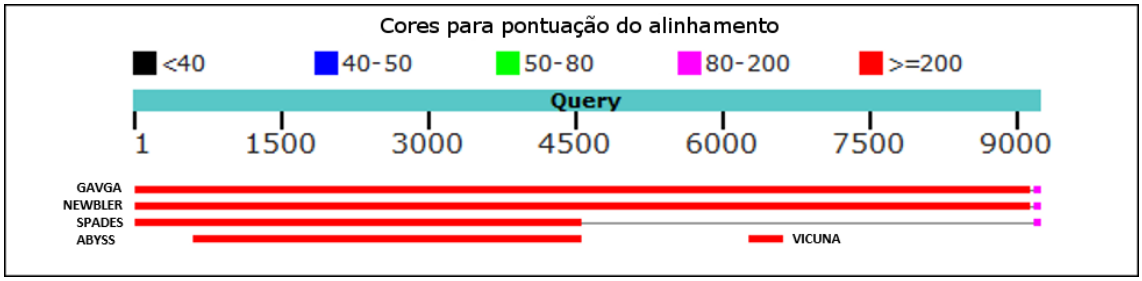
Software Montador	*Total de Contigs*	*Maior Contig (bp)*	*Total de bases (bp)*
GAVGA (tam pop =2)	166,7 ± 4,56	7915,3 ± 1033,59	62990,2 ± 1977,12
GAVGA (tam pop =6)	165,5 ± 3,96	8494,1 ± 596,43	63890,6 ± 3316
GAVGA (tam pop =10)	162,4 ± 6,47	7504,2 ± 832,57	62592,9 ± 4549,56
Newbler	3	9077	9828
SPAdes	131	4543	18001
ABySS	149	3962	18007
VICUNA	143	439	22605

Os resultados para cada montador são mostrados na tabela no que diz respeito ao total de *contigs* gerados, o maior *contig* encontrado e o total de pares de base dos *contigs* finais.

Os resultados da Tabela 1 mostram que com uma população de 6 indivíduos, o GAVGA obteve o maior tamanho médio do maior *contig*. O número de *contigs* obtidos pelo Newbler (3 *contigs*) foram melhores do que os obtidos pelo GAVGA (média de 165 *contigs*) e outros montadores. Por outro lado, o comprimento do maior *contig* obtido pelo GAVGA (9.132 bp), em uma das 10 execuções, teve 55 pares de base a mais que o maior *contig* obtido pelo Newbler (9.077 bp) e consideravelmente maior do que os maiores *contigs* obtidos pelos outros montadores.

A informação genética contida nos *contigs* do GAVGA, assim como a dos outros montadores, estava correta, como mostra a Figura 24, ao se executar um alinhamento dos maiores *contigs* obtidos por cada um dos montadores contra um genoma de HIV1 baixado do NCBI (número de acesso: NC_001802.1), usando a ferramenta BLAST (TATUSOVA; MADDEN, 1999)

Figura 24 - Alinhamento feito no BLAST do maior *contig* obtido por cada software montador contra o banco de HIV1 do NCBI.



Fonte: Elaborada pelo autor, por meio da ferramenta BLAST.

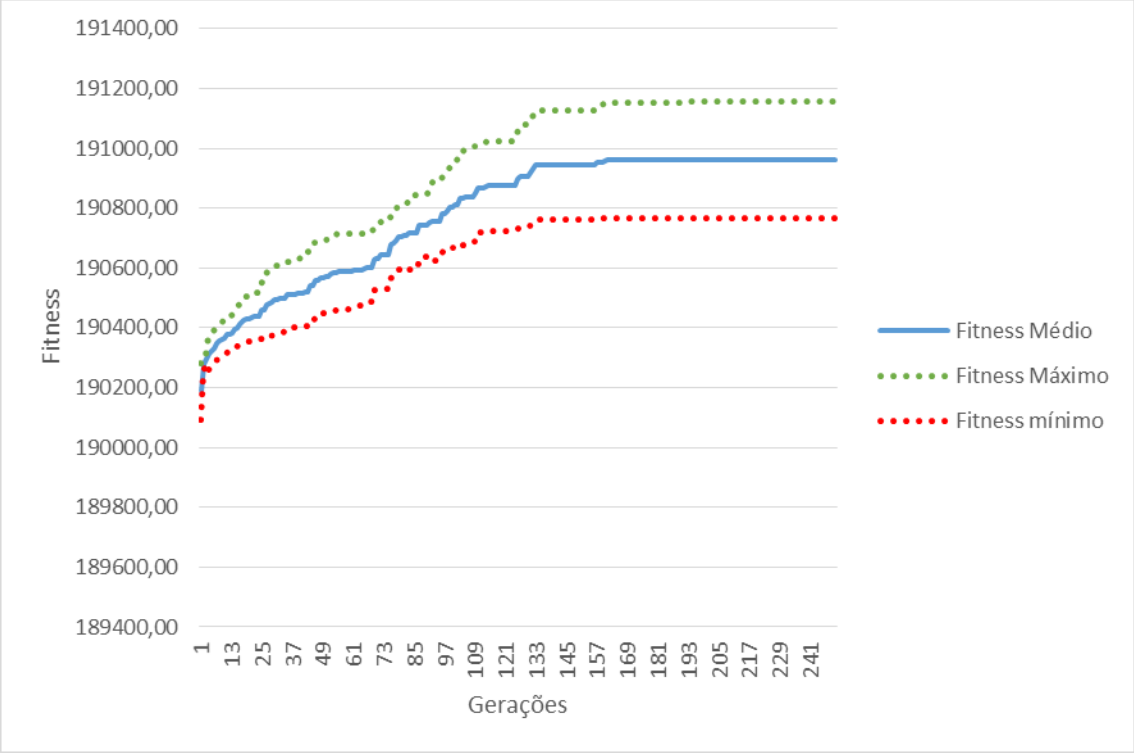
Pelas informações mostradas na Figura 24, verifica-se que o GAVGA montou corretamente as *reads*, uma vez que não houve nenhuma quebra (*gap*) durante o alinhamento e a pontuação de alinhamento e a porcentagem de identidade ficaram altas, como mostra a Tabela 2 abaixo.

Tabela 2 – Pontuação de alinhamento, porcentagem de similaridade e cobertura obtida ao se alinhar os maiores *contigs* de cada ferramenta contra a base de dados do NCBI.

Software Montador	*Pontuação de alinhamento*	*Porcentagem de similaridade (%)*	*Cobertura (%)*
GAVGA	11806	90	99
Newbler	12223	91	99
SPAdes	6832	93	49
ABySS	5829	93	42
VICUNA	379	86	3

Na Figura 25, é mostrada a convergência do GAVGA para obter os melhores *contigs*. Para mostrar a tendência da convergência, foi escolhido analisar a primeira população criada na fase de clusterização do GAVGA com o tamanho da população de 6 indivíduos, cujos indivíduos são compostos de 626 genes.

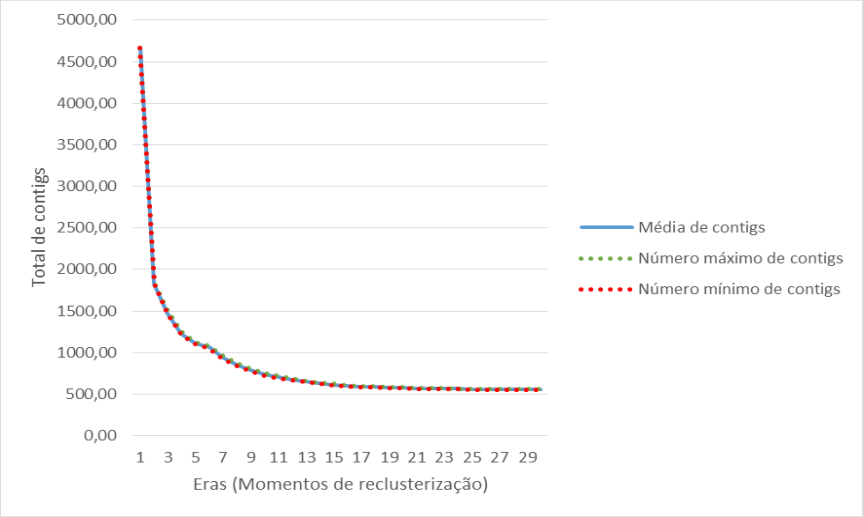
Figura 25 – Média do fitness dos indivíduos calculado em 10 execuções do GAVGA.



Fonte: Elaborada pelo autor.

Quando toda as populações (formadas inicialmente no processo de clusterização) convergem, o GAVGA monta o melhor indivíduo de cada uma delas, gerando então *contigs* que são as melhores soluções encontradas. Tais *contigs* de cada população são reunidos e reclusterizados em novas populações. A Figura 26 mostra o total de *contigs* obtido em cada momento de reclusterização (chamado de “era”) do GAVGA ao montar o conjunto de *reads* reais com uma população de tamanho 6.

Figura 26 – Convergência do total de *contigs* em cada momento de reclusterização feito pelo GAVGA.

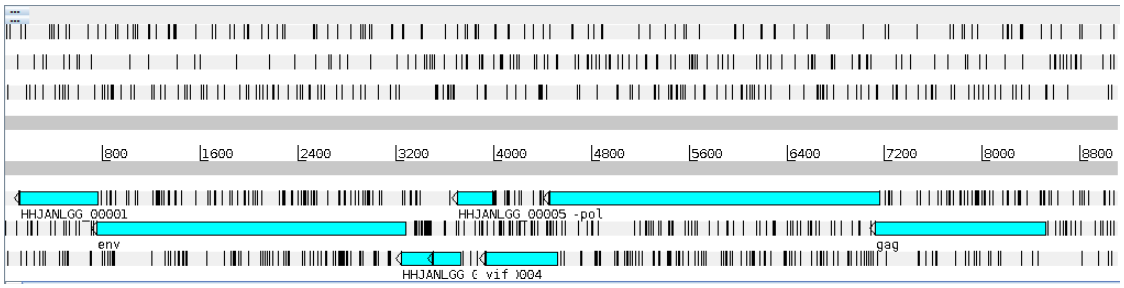


Fonte: Elaborada pelo autor.

É possível observar na Figura 26 que o número de sequências vem sendo minimizado a cada momento de reclusterização sem haver perda de informação de alguma *read*, sendo assim montadas ao longo da execução do algoritmo. Os desvios padrão também são bem pequenos, como mostra a Figura 26, onde os valores máximos e mínimos estão bem próximos da média. Outra forma de se avaliar uma montagem é verificando se os genes presentes nos *contigs* obtidos estão bem montados. Na

Figura 27, por exemplo, temos a anotação de um dos *contigs* montados pelo GAVGA.

Figura 27 - Anotação gerada pelo PROKKA de um *contig* de tamanho 9.108 bp montado pelo GAVGA. Visualização feita na ferramenta Artemis.



Fonte: Elaborada pelo autor, por meio da ferramenta Artemis.

Pela

Figura 27 podemos notar que nesta montagem feita pelo GAVGA não houve fragmentação de genes, ou seja, todos os genes estão completos, sem estarem quebrados em um outro *frame* de leitura.

4.2 CONJUNTO DE *READS* SIMULADAS

A simulação do sequenciamento do vírus da Herpes descrito na seção 3.1 gerou um total de 10.724 *reads* do tipo *single* (não pareadas). Os parâmetros utilizados nas montagens feitas pelo GAVGA e os outros montadores foram os mesmos dos utilizados na montagem das *reads* reais. Os resultados das montagens realizadas podem ser vistos na Tabela 3.

Tabela 3 - Resultados estatísticos do GAVGA, Newbler, SPAdes, ABySS e VICUNA ao montar *reads* simuladas.

Montador	***Total de Contigs***	***Maior contig (bp)***	***Total de bases (bp)***	***Similaridade do maior contig contra a referência (%)***	***Cobertura com relação à referência (%)***
GAVGA (tam pop =2)	31,6 ± 4,88	87405,3 ± 32575,94	231999 ± 23200,99	97,81 ± 0,3	61,27 ± 20,48
GAVGA (tam pop =6)	31,6 ± 3,88	100176,8 ± 28322,80	229566,3 ± 20119,67	98 ± 0	68,1 ± 20,2
GAVGA (tam pop =10)	31,5 ± 4,46	65878,1 ± 11515,60	227887,4 ± 17598,31	97,7 ± 0.6	46,4 ± 8.66
Newbler	3	107952	136387	99	70
SPAdes	38	54680	137498	99	35
ABySS	111	28397	139986	99	18
VICUNA	411	436	63974	100	0.2

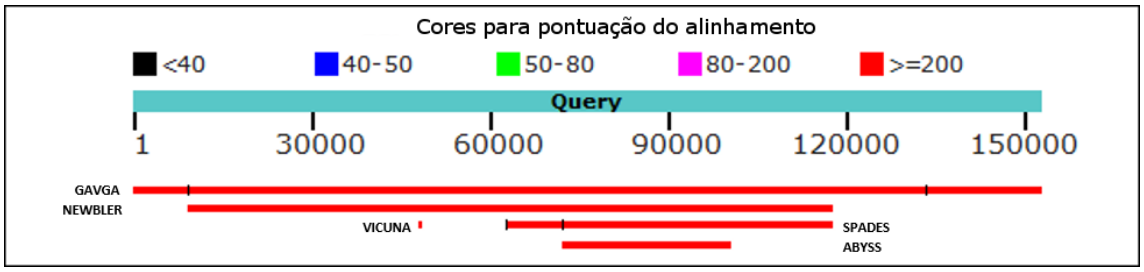
Os resultados para cada montador são mostrados na tabela no que diz respeito ao total de *contigs* gerados, o maior *contig* encontrado e o total de pares de base dos *contigs* finais. Como agora está sendo montado um conjunto de *reads* simuladas, é possível calcular a similaridade do *contig* contra o genoma da referência e a sua cobertura.

O GAVGA também foi executado com 3 tamanhos diferentes de população ao montar o conjunto de *reads* simuladas. Os resultados da Tabela 3 mostram que a

população de 6 indivíduos retornou bons resultados. O número de *contigs* obtidos pelo Newbler (3 *contigs*) foi o menor dentre todos os montadores. O GAVGA obteve uma média do total de *contigs* menor que o SPAdes (38 *contigs*), AbySS (111 *contigs*) e VICUNA (411 *contigs*). Além disso, o comprimento do maior *contig* obtido pelo GAVGA (154.399 bp) em uma das 10 execuções teve apenas 3 bp a mais que o genoma completo do vírus da Herpes usado como referência (154.396 bp) e com 46.447 bp a mais que o maior *contig* obtido pelo Newbler (107.952 bp) e consideravelmente maior que os maiores *contigs* obtidos pelos outros montadores. Nesse caso, quanto maior e mais próximo o tamanho do *contig* for do tamanho da referência e quanto mais próximo de 100% de similaridade com a referência, melhor foi a montagem.

Na Figura 28 tem-se o alinhamento BLAST dos maiores *contigs* obtidos pelo GAVGA e pelos outros montadores contra o genoma de referência (Vírus da Herpes).

Figura 28 - Alinhamento feito no BLAST do maior *contig* obtido por cada software montador contra o genoma de referência (Vírus da Herpes).



Fonte: Elaborada pelo autor, por meio da ferramenta BLAST.

A informação apresentada na Figura 28 mostra que o GAVGA montou as *reads* simuladas corretamente e que obteve o genoma completo do vírus da Herpes, o que mostra que o algoritmo apresenta uma boa confiabilidade ao montar um genoma.

5. CONCLUSÃO

Este trabalho apresentou o GAVGA (*Genetic Algorithm for Viral Genome Assembly*), um algoritmo genético para montagem de genomas virais. O GAVGA foi capaz de montar *reads single-end* e pareadas (ver Seção 3.1), independente da tecnologia de sequenciamento utilizada. Diferentemente dos outros montadores do estado da arte, que utilizam conceitos de grafos para montar as *reads* em um genoma, foi demonstrado como um algoritmo genético também pode ser usado para a mesma finalidade. Os resultados com os conjuntos de dados reais e simulados mostraram que o GAVGA obteve *contigs* maiores e mais próximos do tamanho dos genomas de referência que os *contigs* obtidos pelos outros montadores. Além disso, os *contigs* do GAVGA gerados tanto pelo teste com dados reais quanto pelo teste com dados simulados foram muito similares às suas referências (cerca de 98%), o que mostra que os *contigs* foram montados corretamente.

Em uma futura versão do GAVGA, pretende-se realizar a implementação em CUDA do código, o que irá permitir que o algoritmo processe cada população criada (*cluster* de *reads*) de forma paralela, usando centenas de *threads* simuladas por uma GPU (*Graphics Processing Unit*). Além disso, também pretende-se otimizar os operadores genéticos de forma que também possam ser paralelizados por *threads* de CPU.

5.1 TRABALHOS PUBLICADOS

O artigo referente ao GAVGA foi aceito na 18th *EPIA Conference on Artificial Intelligence*, qualis B1, e publicado pela Springer (OLIVEIRA *et al.*, 2017).

REFERÊNCIAS

ALI, Mostafa Z. *et al.* An improved class of real-coded Genetic Algorithms for numerical optimization ☆. **Neurocomputing** , maio 2017. Disponível em: <http://linkinghub.elsevier.com/retrieve/pii/S0925231217309177>. Acesso em: 14 set. 2017.

BADGER, JH; OLSEN, GJ. CRITICA: coding region identification tool invoking comparative analysis. **Molecular biology and evolution** , 1999. Disponível em: <https://academic.oup.com/mbe/article-abstract/16/4/512/2925441>. Acesso em: 14 set. 2017.

BANKEVICH, Anton *et al.* SPAdes: A New Genome Assembly Algorithm and Its Applications to Single-Cell Sequencing. **Journal of Computational Biology** v. 19, n. 5, p. 455–477 , 7 maio 2012. Disponível em: <http://online.liebertpub.com/doi/abs/10.1089/cmb.2012.0021>. Acesso em: 13 set. 2017.

BATZOGLOU, Serafim *et al.* ARACHNE: a whole-genome shotgun assembler. **Genome research** v. 12, n. 1, p. 177–89 , 1 jan. 2002. Disponível em: <http://www.ncbi.nlm.nih.gov/pubmed/11779843>. Acesso em: 13 set. 2017.

BAYLIS, Charles *et al.* Circuit optimization algorithms for real-time spectrum sharing between radar and communications. maio 2016, [S.l.]: IEEE, maio 2016. p.1–4. Disponível em: <http://ieeexplore.ieee.org/document/7485065/>. Acesso em: 14 set. 2017. 978-1-5090-0863-6. .

BERTOSSI, Alan A. The edge Hamiltonian path problem is NP-complete. **Information Processing Letters** v. 13, n. 4–5, p. 157–159 , jan. 1981. Disponível em: <http://linkinghub.elsevier.com/retrieve/pii/002001908190048X>. Acesso em: 13 set.

2017.

BIOPHYSIK, Promotionsfach :; SUHAI, S. MIRA: An Automated Genome and EST Assembler. , 2007. Disponível em: <http://archiv.ub.uni-heidelberg.de/volltextserver/7871/1/thesis_zusammenfassung.pdf>. Acesso em: 13 set. 2017.

BLICKLE, T. Theory of evolutionary algorithms and application to system synthesis. , 1997.

CARVER, T *et al.* Artemis: an integrated platform for visualization and analysis of high-throughput sequence-based experimental data. , 2011. Disponível em: <https://academic.oup.com/bioinformatics/article-abstract/28/4/464/213043>. Acesso em: 14 set. 2017.

CHU, Justin *et al.* BioBloom tools: fast, accurate and memory-efficient host species sequence screening using bloom filters. **Bioinformatics** v. 30, n. 23, p. 3402–3404 , 1 dez. 2014. Disponível em: <https://academic.oup.com/bioinformatics/article-lookup/doi/10.1093/bioinformatics/btu558>. Acesso em: 12 set. 2017.

CRICK, F.H.C. The origin of the genetic code. **Journal of Molecular Biology** v. 38, n. 3, p. 367–379 , dez. 1968. Disponível em: <http://linkinghub.elsevier.com/retrieve/pii/0022283668903926>. Acesso em: 14 set. 2017.

DOMINGO, Esteban; SHELDON, Julie; PERALES, Celia. Viral quasispecies evolution. **Microbiology and molecular biology reviews : MMBR** v. 76, n. 2, p. 159–216 , 1 jun. 2012. Disponível em: <http://www.ncbi.nlm.nih.gov/pubmed/22688811>. Acesso em: 11 set. 2017.

EDWARDS, Robert A.; ROHWER, Forest. Viral metagenomics. **Nature Reviews Microbiology** v. 3, n. 6, p. 504–512 , 1 jun. 2005.

FLEISCHNER, Herbert. **Eulerian graphs and related topics. Part 1, volume 1**. [S.l.]: North-Holland, 1990.

FRAGA, Joelmo Silva; SILVA, Joelmo. Algoritmos genéticos e o problema de montagem de reads. , 2014. Disponível em: <http://200.129.202.51:8080/jspui/handle/123456789/2164>. Acesso em: 11 set. 2017.

GILMORE, PC *et al.* The Traveling Salesman Problem. A guided tour of combinatorial optimization. **Lawler, Lenstra, Rinnooy Kan (Eds)** , 1986.

GOÉS, Fabiana *et al.* Towards an Ensemble Learning Strategy for Metagenomic Gene Prediction. [S.l.]: Springer, Cham, 2014. p. 17–24. Disponível em: <http://link.springer.com/10.1007/978-3-319-12418-6_3>. Acesso em: 4 set. 2017.

GORDON, D; ABAJIAN, C; GREEN, P. Consed: a graphical tool for sequence finishing. **Genome research** v. 8, n. 3, p. 195–202 , 1 mar. 1998. Disponível em: <http://www.ncbi.nlm.nih.gov/pubmed/9521923>. Acesso em: 13 set. 2017.

GUTHERY, SL; SALISBURY, BA; PUNGLIYA, MS. The structure of common genetic variation in United States populations. **The American Journal of** , 2007. Disponível em: <http://www.sciencedirect.com/science/article/pii/S0002929707637707>. Acesso em: 14 set. 2017.

HÉAM, P.-C.; HUGOT, V.; KOUCHNARENKO, O. The emptiness problem for tree automata with at least one global disequality constraint is NP-hard. **Information Processing Letters** v. 118, p. 6–9 , fev. 2017. Disponível em: <http://linkinghub.elsevier.com/retrieve/pii/S002001901630134X>. Acesso em: 13 set. 2017.

HENN, Matthew R. *et al.* Whole Genome Deep Sequencing of HIV-1 Reveals the Impact of Early Minor Variants Upon Immune Recognition During Acute Infection. **PLoS Pathogens** v. 8, n. 3, p. e1002529 , 8 mar. 2012. Disponível em:

<http://dx.plos.org/10.1371/journal.ppat.1002529>. Acesso em: 11 set. 2017.

HUANG, Xiaoqiu *et al.* PCAP: a whole-genome assembly program. **Genome research** v. 13, n. 9, p. 2164–70 , 1 set. 2003. Disponível em: <http://www.ncbi.nlm.nih.gov/pubmed/12952883>. Acesso em: 13 set. 2017.

HUGHES, J; HOUGHTEN, S. Recentering and restarting genetic algorithm variations for DNA fragment assembly. **in Bioinformatics and …** , 2014. Disponível em: <http://ieeexplore.ieee.org/abstract/document/6845500/>. Acesso em: 14 set. 2017.

HUNT, Martin *et al.* IVA: accurate *de novo* assembly of RNA virus genomes. **Bioinformatics** v. 31, n. 14, p. 2374–2376 , 15 jul. 2015. Disponível em: <https://academic.oup.com/bioinformatics/article-lookup/doi/10.1093/bioinformatics/btv120>. Acesso em: 12 set. 2017.

INDUMATHY, R; MAHESWARI, SU. Nature inspired algorithms to solve DNA fragment assembly problem: A survey. **International Journal of Bioinformatics** , 2012. Disponível em: <http://www.academia.edu/download/36849805/2212ijbb05.pdf>. Acesso em: 14 set. 2017.

JURKIEWICZ, Samuel. Grafos -Uma Introdução. **GrafosModfranc** , 2008. Disponível em: <http://200.17.137.109:8081/novobsi/Members/silvana/matematica-discreta-2o-2015/GrafosModfrancisca.pdf>. Acesso em: 13 set. 2017.

KIKUCHI, S.; CHAKRABORTY, G. Heuristically Tuned GA to Solve Genome Fragment Assembly Problem. 2006, [S.l.]: IEEE, 2006. p.1491–1498. Disponível em: <http://ieeexplore.ieee.org/document/1688485/>. Acesso em: 12 set. 2017. 0-7803-9487-9. .

LEMOS, Melissa; BASÍLIO, Antônio; CASANOVA, Marco Antônio. Um Estudo dos Algoritmos de Montagem de Fragmentos de DNA. , 2003. Disponível em: <http://www.dbd.puc-rio.br/depto_informatica/03_05_lemos.pdf>. Acesso em: 12 set.

2017.

LI, Z. *et al.* Comparison of the two major classes of assembly algorithms: overlap-layout-consensus and de-bruijn-graph. **Briefings in Functional Genomics** v. 11, n. 1, p. 25–37 , 1 jan. 2012. Disponível em: <https://academic.oup.com/bfg/article-lookup/doi/10.1093/bfgp/elr035>. Acesso em: 4 set. 2017.

LUKE, S. Essentials of metaheuristics. , 2009. Disponível em: <http://www.cs.put.poznan.pl/mkomosinski/materialy/optymalizacja/extras/Essentials.pdf>. Acesso em: 14 set. 2017.

MANGUL, S. *et al.* Accurate viral population assembly from ultra-deep sequencing data. **Bioinformatics** v. 30, n. 12, p. i329–i337 , 15 jun. 2014. Disponível em: <https://academic.oup.com/bioinformatics/article-lookup/doi/10.1093/bioinformatics/btu295>. Acesso em: 12 set. 2017.

METZKER, Michael L. Sequencing technologies — the next generation. **Nature Reviews Genetics** v. 11 , 2009. Disponível em: <http://biology.umd.edu/uploads/2/7/8/0/27804901/metzker_2012.pdf>. Acesso em: 13 set. 2017.

MODJARRAD, Kayvon; KOFF, Wayne C. **Human vaccines : emerging technologies in design and development**. [S.l.]: Academic Press, 2017.

MONTAZERI-GH, Morteza; POURSAMAD, Amir; GHALICHI, Babak. Application of genetic algorithm for optimization of control strategy in parallel hybrid electric vehicles. **Journal of the Franklin Institute** v. 343, n. 4–5, p. 420–435 , jul. 2006. Disponível em: <http://linkinghub.elsevier.com/retrieve/pii/S0016003206000329>. Acesso em: 14 set. 2017.

MYERS, Eugene W. *et al.* A Whole-Genome Assembly of Drosophila. **Science** v. 287, n. 5461 , 2000. Disponível em: <http://science.sciencemag.org/content/287/5461/2196>.

Acesso em: 13 set. 2017.

MYERS JR, Eugene W. A history of DNA sequence assembly. **it - Information Technology** v. 58, n. 3, p. 126–132 , 28 jan. 2016. Disponível em: <https://www.degruyter.com/view/j/itit.2016.58.issue-3/itit-2015-0047/itit-2015-0047.xml>. Acesso em: 13 set. 2017.

NAGARAJAN, Niranjan; POP, Mihai. Sequence assembly demystified. **Nature Reviews Genetics** v. 14, n. 3, p. 157–167 , 29 jan. 2013. Disponível em: <http://www.nature.com/doifinder/10.1038/nrg3367>. Acesso em: 4 set. 2017.

NAKAMURA, Yoji *et al.* V-GAP: Viral genome assembly pipeline. **Gene** v. 576, n. 2, p. 676–680 , 2016. Disponível em: <http://www.sciencedirect.com/science/article/pii/S0378111915012378>. Acesso em: 12 set. 2017.

NEGNEVITSKY, M. Artificial intelligence: a guide to intelligent systems. , 2005.

NEUMANN, Avidan U. *et al.* Hepatitis C Viral Dynamics in Vivo and the Antiviral Efficacy of Interferon-α Therapy. **Science** v. 282, n. 5386 , 1998.

NEUMANN, Frank; WITT, Carsten. Bioinspired computation in combinatorial optimization. 2013, New York, New York, USA: ACM Press, 2013. p.567. Disponível em: <http://dl.acm.org/citation.cfm?doid=2464576.2466738>. Acesso em: 14 set. 2017. 9781450319645. .

OLIVEIRA, Renato R. M. *et al.* GAVGA: A Genetic Algorithm for Viral Genome Assembly. [S.l.]: Springer, Cham, 2017. p. 395–407. Disponível em: <http://link.springer.com/10.1007/978-3-319-65340-2_33>. Acesso em: 12 set. 2017.

PACHECO, MAC. Algoritmos genéticos: princípios e aplicações. **Elétrica. Pontifícia Universidade Católica do Rio …** , 1999. Disponível em: <http://www2.ica.ele.puc-

rio.br/Downloads%5C38/CE-Apostila-Comp-Evol.pdf>. Acesso em: 14 set. 2017.

PAEZ-ESPINO, David *et al.* Uncovering Earth's virome. **Nature** v. 536, n. 7617, p. 425–430 , 17 ago. 2016. Disponível em: <http://www.nature.com/doifinder/10.1038/nature19094>. Acesso em: 14 set. 2017.

PALMER, Lance E *et al.* Improving de novo sequence assembly using machine learning and comparative genomics for overlap correction. **BMC Bioinformatics** v. 11, n. 1, p. 33 , 2010. Disponível em: <http://bmcbioinformatics.biomedcentral.com/articles/10.1186/1471-2105-11-33>. Acesso em: 4 set. 2017.

PARSONS, Lance R *et al.* Rapid genome assembly and comparison decode intrastrain variation in human alphaherpesviruses. **mBio** v. 6, n. 2, p. e02213-14 , 31 mar. 2015. Disponível em: <http://www.ncbi.nlm.nih.gov/pubmed/25827418>. Acesso em: 12 set. 2017.

PARSONS, Rebecca J.; FORREST, Stephanie; BURKS, Christian. Genetic algorithms, operators, and DNA fragment assembly. **Machine Learning** v. 21, n. 1–2, p. 11–33 , 1995. Disponível em: <http://link.springer.com/10.1007/BF00993377>. Acesso em: 11 set. 2017.

PEREIRA, Vivian M Y; COSTA, Camila I; DIGIAMPIETRI, Luciano A. Uma ferramenta baseada em algoritmos genéticos para a ordena ao de montagens parciais de genomas. , 2014.

PEVSNER, Jonathan. **Bioinformatics and functional genomics**. [S.l: s.n.], 2015. 1124 p.

PEVZNER, P A; TANG, H; WATERMAN, M S. An Eulerian path approach to DNA fragment assembly. **Proceedings of the National Academy of Sciences of the United States of America** v. 98, n. 17, p. 9748–53 , 14 ago. 2001. Disponível em:

<http://www.ncbi.nlm.nih.gov/pubmed/11504945>. Acesso em: 13 set. 2017.

POIRIER, Enzo Z; VIGNUZZI, Marco. Virus population dynamics during infection. **Current Opinion in Virology** v. 23, p. 82–87 , abr. 2017. Disponível em: <http://linkinghub.elsevier.com/retrieve/pii/S1879625716302085>. Acesso em: 11 set. 2017.

PRANGISHVILI, D; GARRETT, RA; KOONIN, EV. Evolutionary genomics of archaeal viruses: unique viral genomes in the third domain of life. **Virus research** , 2006. Disponível em: <http://www.sciencedirect.com/science/article/pii/S0168170206000268>. Acesso em: 14 set. 2017.

RECHENBERG, I. Evolution Strategy: Optimization of Technical systems by means of biological evolution. **Fromman-Holzboog, Stuttgart** , 1973. Disponível em: <https://scholar.google.com.br/scholar?q=Evolution+Strategy%3A+Optimization+of+Technical+systems+by+means+of+biological+evolution&btnG=&hl=pt-BR&as_sdt=0%2C5>. Acesso em: 14 set. 2017.

RIBEIRO, Paulo Henrique *et al.* Fundamentos de Algoritmos Evolutivos. , 2008. Disponível em: <https://pdfs.semanticscholar.org/c93f/56a62a3da7e704f67831cb845bd326808d82.pdf>. Acesso em: 14 set. 2017.

RICHARDSON, EJ; WATSON, M. The automatic annotation of bacterial genomes. **Briefings in bioinformatics** , 2012. Disponível em: <https://academic.oup.com/bib/article-abstract/14/1/1/304727>. Acesso em: 14 set. 2017.

RUBY, J. Graham; BELLARE, Priya; DERISI, Joseph L. PRICE: Software for the Targeted Assembly of Components of (Meta) Genomic Sequence Data. **G3: Genes, Genomes, Genetics** v. 3, n. 5 , 2013. Disponível em:

<http://www.g3journal.org/content/3/5/865.short>. Acesso em: 12 set. 2017.

SANJUÁN, Rafael *et al.* Viral mutation rates. **Journal of virology** v. 84, n. 19, p. 9733–48 , 1 out. 2010. Disponível em: <http://www.ncbi.nlm.nih.gov/pubmed/20660197>. Acesso em: 8 nov. 2017.

SCHMIEDER, R.; EDWARDS, R. Quality control and preprocessing of metagenomic datasets. **Bioinformatics** v. 27, n. 6, p. 863–864 , 15 mar. 2011. Disponível em: <https://academic.oup.com/bioinformatics/article-lookup/doi/10.1093/bioinformatics/btr026>. Acesso em: 12 set. 2017.

SEEMANN, T. Prokka: rapid prokaryotic genome annotation. **Bioinformatics** , 2014. Disponível em: <https://academic.oup.com/bioinformatics/article-abstract/30/14/2068/2390517>. Acesso em: 14 set. 2017.

SENTHILKUMAR, M *et al.* A Modified and Efficient Genetic Algorithm to Address a Travelling Salesman Problem. **International Journal of Applied Engineering Research ISSN** v. 9, n. 10, p. 973–4562 , 2014. Disponível em: <http://www.ripublication.com>. Acesso em: 14 set. 2017.

SIMPSON, Jared T *et al.* ABySS: a parallel assembler for short read sequence data. **Genome research** v. 19, n. 6, p. 1117–23 , 1 jun. 2009. Disponível em: <http://www.ncbi.nlm.nih.gov/pubmed/19251739>. Acesso em: 13 set. 2017.

STANAWAY, Jeffrey D *et al.* The global burden of viral hepatitis from 1990 to 2013: findings from the Global Burden of Disease Study 2013. **The Lancet** v. 388, n. 10049, p. 1081–1088 , set. 2016. Disponível em: <http://linkinghub.elsevier.com/retrieve/pii/S0140673616305797>. Acesso em: 11 set. 2017.

STERKY, Fredrik; LUNDEBERG, Joakim. Sequence analysis of genes and genomes. **Journal of Biotechnology** v. 76, n. 1, p. 1–31 , 2000. Disponível em:

<http://www.sciencedirect.com/science/article/pii/S0168165699001765 %5B2>. Acesso em: 13 set. 2017.

TATUSOVA, Tatiana A; MADDEN, Thomas L. BLAST 2 Sequences, a new tool for comparing protein and nucleotide sequences. **FEMS Microbiology Letters** v. 174, n. 2, p. 247–250 , 1 maio 1999. Disponível em: <https://academic.oup.com/femsle/article-lookup/doi/10.1111/j.1574-6968.1999.tb13575.x>. Acesso em: 6 set. 2017.

TCHEREPANOV, V; EHLERS, A. Genome Annotation Transfer Utility (GATU): rapid annotation of viral genomes using a closely related reference genome. **BMC** , 2006. Disponível em: <https://bmcgenomics.biomedcentral.com/articles/10.1186/1471-2164-7-150>. Acesso em: 14 set. 2017.

WAN, Yinan *et al.* VirAmp: a galaxy-based viral genome assembly pipeline. **GigaScience** v. 4, n. 1, p. 19 , 28 dez. 2015. Disponível em: <https://academic.oup.com/gigascience/article-lookup/doi/10.1186/s13742-015-0060-y>. Acesso em: 12 set. 2017.

WANG, S; SUNDARAM, JP; SPIRO, D. VIGOR, an annotation program for small viral genomes. **BMC** , 2010. Disponível em: <https://bmcbioinformatics.biomedcentral.com/articles/10.1186/1471-2105-11-451>. Acesso em: 14 set. 2017.

WEISS, Vinicius Almir. Estratégias de finalização da montagem do genoma da Bactéria Diazotrófica Endofítica Herbaspirillum Seropedicae SmR1. , 2010. Disponível em: <http://www.acervodigital.ufpr.br/handle/1884/23540>. Acesso em: 13 set. 2017.

YAMASHITA, Akifumi; SEKIZUKA, Tsuyoshi; KURODA, Makoto. VirusTAP: Viral Genome-Targeted Assembly Pipeline. **Frontiers in microbiology** v. 7, p. 32 , 2016. Disponível em: <http://www.ncbi.nlm.nih.gov/pubmed/26870004>. Acesso em: 12 set. 2017.

YANG, Xiao *et al.* De novo assembly of highly diverse viral populations. **BMC Genomics** v. 13, n. 1, p. 475 , 2012. Disponível em: <http://bmcgenomics.biomedcentral.com/articles/10.1186/1471-2164-13-475>. Acesso em: 12 set. 2017.

YANG, Xiao; ZOLA, Jaroslaw; ALURU, Srinivas. Parallel Metagenomic Sequence Clustering Via Sketching and Maximal Quasi-clique Enumeration on Map-Reduce Clouds. maio 2011, [S.l.]: IEEE, maio 2011. p.1223–1233. Disponível em: <http://ieeexplore.ieee.org/document/6012859/>. Acesso em: 12 set. 2017. 978-1-61284-372-8. .

ZERBINO, DR; BIRNEY, E. Velvet: algorithms for de novo short read assembly using de Bruijn graphs. **Genome research** , 2008. Disponível em: <http://genome.cshlp.org/content/18/5/821.short>. Acesso em: 14 set. 2017.

ZUBEN, FJ Von. Computação evolutiva: uma abordagem pragmática. **Tutorial: Notas de Aula da disciplina** , 2000. Disponível em: <ftp://calhau.dca.fee.unicamp.br/pub/docs/vonzuben/ia013_1s07/tutorialEC.pdf>. Acesso em: 14 set. 2017.