UNIVERSIDADE FEDERAL DO PARÁ
INSTITUTO DE CIÊNCIAS EXATAS E NATURAIS
FACULDADE DE COMPUTAÇÃO
CURSO DE BACHARELADO EM CIÊNCIA DA COMPUTAÇÃO

FÁBIO MALCHER MIRANDA

DESENVOLVIMENTO DO PROGRAMA DE COMPUTADOR GAPBLASTER: UMA FERRAMENTA GRÁFICA PARA FECHAMENTO DE GAPS EM GENOMAS PROCARIOTOS

Belém
2017

UNIVERSIDADE FEDERAL DO PARÁ
INSTITUTO DE CIÊNCIAS EXATAS E NATURAIS
FACULDADE DE COMPUTAÇÃO
CURSO DE BACHARELADO EM CIÊNCIA DA COMPUTAÇÃO

FÁBIO MALCHER MIRANDA

DESENVOLVIMENTO DO PROGRAMA DE COMPUTADOR GAPBLASTER: UMA FERRAMENTA GRÁFICA PARA FECHAMENTO DE GAPS EM GENOMAS PROCARIOTOS

Trabalho de Conclusão de Curso apresentado para obtenção do grau de Bacharel em Ciência da Computação.

Orientador: Prof. Dr. Rommel Thiago Juca Ramos

Belém
2017

Malcher Miranda, Fábio
Desenvolvimento do Programa de Computador GapBlaster: uma ferramenta gráfica para fechamento de gaps em genomas procariotos/ Fábio Malcher Miranda. – Belém, 2017.
74 p. : il. (algumas color.) ; 30 cm.

Orientador: Prof. Dr. Rommel Thiago Juca Ramos

Monografia – Universidade Federal do Pará
Instituto de Ciências Exatas e Naturais
Curso de Bacharelado em Ciência da Computação, 2017.

1. Bioinformática. 2. Curadoria de genomas. 3. Fechamento de gaps. I. Título.

FÁBIO MALCHER MIRANDA

DESENVOLVIMENTO DO PROGRAMA DE COMPUTADOR GAPBLASTER: UMA FERRAMENTA GRÁFICA PARA FECHAMENTO DE GAPS EM GENOMAS PROCARIOTOS

Trabalho de Conclusão de Curso apresentado para obtenção do grau de Bacharel em Ciência da Computação.

Data da Defesa: 09 de Março de 2017
Conceito: Excelente

Banca Examinadora

Prof. Dr. Rommel Thiago Juca Ramos
Faculdade de Biotecnologia - UFPA
Orientador

Prof. Dr. Nelson Cruz Sampaio Neto
Faculdade de Computação - UFPA
Membro da Banca

Prof. Dra. Adriana Ribeiro Carneiro
Faculdade de Biotecnologia - UFPA
Membro da Banca

Belém
2017

Dedico este trabalho à minha mãe, que sempre apoiou meus estudos e não mediu esforços para que eu chegasse até aqui.

AGRADECIMENTOS

Os agradecimentos principais são direcionados ao Professor Rommel Ramos, pela excelente orientação e pela paciência e atenção despendidas durante a confecção desta monografia, além de ter me dado a oportunidade de aprender mais sobre o processo de curadoria de genomas, uma etapa importante no trabalho de pesquisadores da área de bioinformática.

Agradeço também ao Professor Artur Luiz e ao CNPq, pela concessão das bolsas de iniciação científica que permitiram o desenvolvimento deste projeto, e ao colega de laboratório Mateus Pinto, que me ajudou a nomear o aplicativo.

Aos professores da Faculdade de Computação, por todas as lições técnicas e de vida que aprendi ao longo destes últimos cinco anos. Sou muito grato também ao meu irmão Felipe, que me ajudou durante o processo de concessão do visto para o intercâmbio acadêmico.

Por último, gostaria de reconhecer o formidável trabalho realizado pela equipe de desenvolvimento do *abnTEX2*[1], cujos esforços facilitaram imensamente a formatação desta monografia dentro dos padrões definidos pela ABNT.

[1] Disponível através do endereço <http://www.abntex.net.br>

"A verdadeira viagem de descobrimento
não consiste em procurar novas paisagens,
mas em ter novos olhos."
(Marcel Proust)

RESUMO

Entre os principais desafios que impedem cientistas de usufruírem dos benefícios decorrentes da montagem de genomas estão os altos custos experimentais, que dificultam a finalização de montagens. Embora existam inúmeras soluções computacionais que auxiliem na etapa de finalização, estas ferramentas são basicamente "caixas pretas" que não apresentam informações suficientes sobre as alterações que estão sendo feitas no genoma e frequentemente não conseguem preencher todas as lacunas deixadas pelos montadores. Neste trabalho, esses problemas são abordados através da criação do programa de computador *GapBlaster*, uma ferramenta gráfica que tem como objetivo reduzir a fragmentação da montagem de genomas e agilizar a finalização desta. O algoritmo proposto utiliza como entrada um arquivo *FASTA* contendo os *scaffolds* do genoma e um ou mais arquivos *FASTA* contendo os *contigs*, que podem ser provenientes de diferentes montadores ou mesmo de outros experimentos de sequenciamento. Os *contigs* são então alinhados contra os *scaffolds* através do programa *Legacy BLAST*, *BLAST+* ou *MUMmer* e os alinhamentos identificados pertencentes a um mesmo *contig* e que podem fechar *gaps* são então exibidos ao usuário para inspeção através da interface gráfica. O desempenho do *GapBlaster* foi avaliado em dados NGS obtidos a partir da montagem das bactérias *Staphylococcus aureus* e *Rhodobacter sphaeroides*, ambas adquiridas do projeto GAGE. Adicionalmente, foram utilizados dados NGS da montagem do genoma *Corynebacterium pseudotuberculosis*. Comparado com ferramentas similares, os resultados do *GapBlaster* são satisfatórios e ele apresenta a vantagem de oferecer uma interface gráfica para inspeção dos *gaps* fechados. O *GapBlaster* foi implementado na linguagem de programação *Java* e está disponível para *download* através do endereço <https://sourceforge.net/projects/gapblaster/>. O *GapBlaster* requer a instalação do *JDK 8* e do *Legacy BLAST*, *BLAST+* ou *MUMmer*.

Palavras–chave: bioinformática. curadoria de genomas. fechamento de *gaps*.

ABSTRACT

Among the main challenges that prevent scientists from reaping the benefits of genome assembly are the high experimental costs, which make it difficult to finish assemblies. Although there are various computational solutions that assist in the finishing phase, these tools are basically "black boxes" that do not present enough information about the changes being made in the genome and often fail to fill in all the gaps left by the assemblers. In this work, these issues are addressed through the creation of the software GapBlaster, a graphical tool that aims to reduce the fragmentation of genomes assembly and speed-up finishing the latter. The proposed algorithm uses as input a FASTA file containing the scaffolds of the genome and one or more FASTA files containing the contigs, which may come from different assemblers or even from other sequencing experiments. The contigs are then aligned against the scaffolds through the software Legacy BLAST, BLAST + or MUMmer and the identified alignments belonging to the same contig that can also close gaps are then displayed to the user for inspection through the graphical user interface. GapBlaster's performance was evaluated on NGS data obtained from the assemblies of the bacterias *Staphylococcus aureus* and *Rhodobacter sphaeroides*, both acquired from the GAGE project. Additionally, NGS data from the genome assembly of *Corynebacterium pseudotuberculosis* was used. Compared with similar tools, the results of GapBlaster are satisfactory and it has the advantage of offering a graphical user interface for inspection of closed gaps. GapBlaster was developed in Java and is available for download at <https://sourceforge.net/projects/gapblaster/>. GapBlaster requires the installation of JDK 8 and Legacy BLAST, BLAST + or MUMmer.

Keywords: bioinformatics. genome curation. gap closing.

LISTA DE ILUSTRAÇÕES

LISTA DE QUADROS

LISTA DE TABELAS

LISTA DE ALGORITMOS

LISTA DE ABREVIATURAS E SIGLAS

ABySS	*Assembly By Short Sequences*
BLAST	*Basic Local Alignment Search Tool*
CABOG	*Celera Assembler with Best Overlap Graph*
DNA	*Deoxyribonucleic Acid*
G4ALL	*Graphical Contig Analyzer for All Sequencing Platforms*
GAGE	*Genome Assembly Gold-standard Evaluations*
GOLD	*Genomes Online Database*
IDE	*Integrated Development Environment*
JDK	*Java Development Kit*
MSR-CA	*Maryland Super Read Cabog Assembler*
MUMmer	*Maximal Unique Matcher*
NCBI	*National Center for Biotechnology Information*
NGS	*Next Generation Sequencing*
PB	Pares de Bases
PGM	*Personal Genome Machine*
SGA	*String Graph Assembler*
SNP	*Single Nucleotide Polymorphism*
snRNA	*Small Nuclear Ribonucleic Acid*
SOAP	*Short Oligonucleotide Analysis Package*
SPAdes	*St. Petersburg Genome Assembler*
SRA	*Sequence Read Archive*

LISTA DE SÍMBOLOS

A	Adenina
C	Citosina
G	Guanina
T	Timina

SUMÁRIO

1 INTRODUÇÃO

1.1 Contexto

O surgimento da tecnologia NGS (*Next Generation Sequencing*) trouxe avanços sem precedentes para a biologia, possibilitando importantes descobertas científicas através da redução de custos e do tempo necessário para o sequenciamento de genomas (SCHUSTER, 2007).

Contudo, os sequenciadores NGS apresentam erros inerentes de leitura, dentre os quais podemos citar a inserção, deleção ou substituição de bases (WOJCIESZEK et al., 2014). Por este motivo, os dados obtidos a partir de plataformas NGS devem passar por um pré-processamento, com o objetivo de corrigir erros e selecionar as leituras com melhor qualidade.

Em seguida, os dados filtrados são repassados para um montador, que é responsável por agregar as leituras de maneira a formarem *contigs* e ultimamente ordenar os *contigs* em *scaffolds*. Baker (2012) descreve o processo de montagem de genomas como sendo análogo à montagem de um quebra-cabeça, onde cada leitura seria uma peça do jogo e o resultado final após encaixar cada peça seria a informação genética de um organismo.

Os principais desafios que podem aparecer durante a montagem de genomas são erros de clonagem e ambiguidades em regiões de repetição, que forçam o montador a gerar saídas fragmentadas (NAGARAJAN et al., 2010). A fragmentação de genomas ocorre quando os nucleotídeos, representados *in silico* pelas letras A (Adenina), T (Timina), C (Citosina) ou G (Guanina), não podem ser determinados pelo montador. Estas regiões são denominadas de *gaps* e geralmente são representadas por Ns.

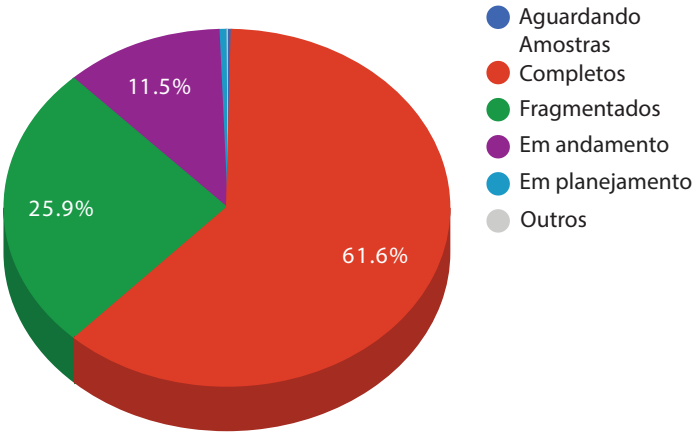
A fim de diminuir o tamanho e quantidade de *gaps*, montadores específicos, que utilizam técnicas de grafos gulosos, grafos de sobreposição e grafos *de bruijn*, foram desenvolvidos para tentar resolver os problemas oriundos dos sequenciadores NGS (MILLER; KOREN; SUTTON, 2010). Porém, mesmo estes montadores não conseguem gerar uma montagem perfeita e torna-se necessária mais uma etapa conhecida como finalização, que tem como objetivo melhorar a qualidade da montagem e consequentemente completar um projeto de sequenciamento.

Apesar dos recentes avanços nos protocolos de montagem e curadoria de genomas, cientistas regularmente ainda se deparam com uma certa dificuldade em finalizá-los. Segundo Mukherjee et al. (2016), cerca de 26% dos projetos de sequenciamento de genomas bacterianos cadastrados no banco de dados GOLD (*Genomes Online Database*) ainda apresentam montagens fragmentadas, conforme ilustra o Gráfico 1.

Um dos principais motivos para que alguns projetos de sequenciamento de genomas não sejam completados é o aumento dos custos provenientes da etapa de finalização *in vitro*, que requer um maior trabalho experimental e pode tornar a completude do projeto impossível para pequenos laboratórios (SOUEIDAN et al., 2013). Soueidan et al. (2013) afirma que o custo do

projeto é elevado devido a finalização *in vitro* requerer um novo sequenciamento de algumas áreas do genoma, com o propósito de resolver problemas de montagem, fechar *gaps* e aumentar a cobertura e qualidade em algumas regiões do genoma. Por esta razão, é interessante que se finalize o máximo possível do genoma utilizando técnicas *in silico* para reduzir custos.

Gráfico 1 – Estado do sequenciamento de genomas bacterianos



Fonte: Mukherjee et al. (2016)

Diversas ferramentas comerciais, tais como os programas *CLC Genomics Workbench* e *Lasergene Suite*, além de *softwares* de código aberto, como o *G4ALL*, *Gap-Closer*, *GapFiller* e *FGAP* foram desenvolvidos para ajudar na etapa de finalização de genomas *in silico*. As estratégias adotadas por estes programas variam desde a utilização de leituras pareadas até a utilização dos próprios dados resultantes de montagens para preencher os *gaps*. Contudo, estes programas não conseguem resolver todas as lacunas deixadas pelos montadores e frequentemente não apresentam informações a respeito das alterações que estão sendo feitas no genoma. Dessa forma, este trabalho propõe o desenvolvimento de uma ferramenta gráfica para fechamento de *gaps in silico*, que permita ao usuário selecionar quais alterações serão feitas no genoma.

1.2 Justificativa

Genomas fragmentados podem dificultar análises posteriores, pois eles apenas representam parcialmente o repertório genético de um organismo e frequentemente contém falhas. De acordo com Fraser et al. (2002), apesar dos aumentos de custos resultantes da etapa de finalização, é importante que sejam produzidos genomas completos, tendo em vista que a ordem e exatidão de todos os pares de base somente é verificada na etapa de finalização de genomas. Por outro lado, Fraser et al. (2002) argumenta que genomas fragmentados podem conter erros de sequenciamento e de montagem, mesmo que possuam alta cobertura.

Estas afirmações ficam evidentes no trabalho publicado por Selkov et al. (2000), cujas análises de genomas microbianos demonstraram que a taxa de erro em genomas finalizados era

de apenas 1 em cada 10000 pares de bases, enquanto que em genomas fragmentados a taxa de erro aumentava para 1 em cada 1000–2000 pares de bases.

Além disso, genomas fragmentados podem prejudicar estudos subsequentes de genômica comparativa e funcional, já que se torna difícil identificar sequências contaminadas até que o projeto de sequenciamento de um genoma seja finalizado. Segundo Fraser et al. (2002), essa contaminação varia entre 5% e 10% do total de leituras sequenciadas de micróbios cultivados em células animais.

Em síntese, é importante que as montagens de genomas sejam finalizadas, para que estudos de genômica comparativa e funcional caracterizem variações de estrutura genômica e conteúdo genético corretamente. Por este motivo, foi desenvolvido neste trabalho o programa de computador *GapBlaster*, que é uma ferramenta computacional com interface gráfica, cujo objetivo é auxiliar na redução de *gaps* através de curadoria manual. Desta forma, é esperado que ocorra aumento na precisão da montagem de genomas, já que não será necessário depender da completa automação da tarefa.

1.3 Objetivos

1.3.1 Objetivo Geral

Criar uma ferramenta computacional que auxilie no fechamento da montagem de genomas bacterianos.

1.3.2 Objetivos Específicos

- Procurar programas que façam o alinhamento de *contigs* contra *scaffolds* de maneira eficiente;
- Estabelecer os melhores algoritmos de alinhamento a serem utilizados;
- Criar a interface gráfica que permita realizar a seleção do alinhador padrão;
- Desenvolver conversores que padronizem os dados gerados pelos alinhadores para um formato único utilizado pelo *GapBlaster*;
- Implementar uma rotina que identifique os alinhamentos que possam ser utilizados para fechar *gaps* e selecione os melhores resultados;
- Criar a interface gráfica que exibirá os resultados para o usuário e permitirá realizar curadoria manual;
- Definir o formato do relatório que contém as informações sobre alterações realizadas;
- Comparar qualitativamente os resultados obtidos com ferramentas similares.

1.4 Metodologia

1.4.1 Ambiente de Desenvolvimento

O *GapBlaster* foi desenvolvido através do paradigma de orientação a objetos na linguagem de programação *Java*[1]. A programação toda foi feita na IDE *NetBeans* 8.1[2] e o código fonte foi compilado com o JDK 8[3]. Também foi utilizada a biblioteca gráfica *Swing*[4] em conjunto com a biblioteca de estilização *BeautyEye* 3.5[5] para criar os recursos visuais e foi usado o repositório de códigos SourceForge[6], com o git[7] como controle de versionamento.

1.4.2 Dados de Teste

O *GapBlaster* foi testado com dois conjuntos de dados de entrada. O primeiro possui dados de montagem do genoma *Corynebacterium pseudotuberculosis* e o segundo contém dados de montagem dos organismos *Staphylococcus aureus* e *Rhodobacter sphaeroides*, obtidos através do projeto GAGE (SALZBERG et al., 2012).

De acordo com Dorella et al. (2006), a *Corynebacterium pseudotuberculosis* é uma bactéria Gram-positiva causadora da doença infecciosa linfadenite caseosa, que acomete pequenos ruminantes, é de difícil tratamento e acarreta prejuízos econômicos em nível global para a indústria agropecuária.

O sequenciamento da bactéria *Corynebacterium pseudotuberculosis* foi realizado na plataforma *Ion Torrent PGM*, conforme demonstra a Tabela 1. As leituras, que estão disponíveis no banco de dados SRA do NCBI sob o registro de número SRR3312980, foram montadas no programa *SPAdes* 3.1.0 usando uma estratégia *de novo* e foram utilizados os parâmetros padrões do *SPAdes* para dados gerados pela plataforma *Ion Torrent PGM*. Os arquivos contendo os *contigs* e os *scaffolds* produzidos pelos *SPAdes* foram utilizados como entrada no *GapBlaster*.

Através do projeto GAGE foram obtidos os arquivos contendo os *contigs* e os *scaffolds* da montagem dos genomas *Staphylococcus aureus* e *Rhodobacter sphaeroides*. O montador *CABOG* foi utilizado somente para a bactéria *Rhodobacter sphaeroides* e os montadores *Abyss*, *ABySS2*, *AllPaths-LG*, *Bambus2*, *MSR-CA*, *SGA*, *SOAPdenovo* e *Velvet* foram utilizados para ambos organismos (SALZBERG et al., 2012). As informações sobre o sequenciamento destes genomas também podem ser vistas na Tabela 1.

[1] Para maiores informações, acesse: <http://java.sun.com>
[2] Disponível através do endereço: <https://netbeans.org/downloads/>
[3] Disponível através do endereço: <http://www.oracle.com/technetwork/java/javase/downloads/>
[4] Para maiores informações, acesse: <http://java.sun.com/docs/books/tutorial/uiswing>
[5] Disponível através do endereço: <https://code.google.com/archive/p/beautyeye/>
[6] Para maiores informações, acesse: <https://sourceforge.net/>
[7] Para maiores informações, acesse: <https://git-scm.com/>

Tabela 1 – Dados de sequenciamento dos genomas utilizados na análise

Organismo	Plataforma	Biblioteca	Tamanho das Leituras	Inserção	Nº de Leituras
C. pseudotuberculosis 262	*Ion PGM*	Fragmentos	∼220 pb	–	1765213
S. aureus A–S391_USA300	*Illumina*	Pares Interligados	∼101 pb	180 pb	1294104
S. aureus A–S391_USA300	*Illumina*	Pares	∼37 pb	3500 pb	3494070
R. sphaeroides 2.4.1	*Illumina*	Pares Interligados	∼101 pb	180 pb	2050868
R. sphaeroides 2.4.1	*Illumina*	Pares	∼101 pb	3500 pb	2050868

Fonte: Sá et al. (2016)

1.4.3 Parâmetros de Entrada

O *GapBlaster* foi implementado para utilizar como entrada arquivos no formato *FASTA*, procedentes da montagem de cada organismo, sendo que um dos arquivos contém os *scaffolds* e os demais contém os *contigs*. De acordo com Mount (2004), o formato *FASTA* é bastante utilizado na bioinformática para prover informações sobre as sequências de nucleotídeos, onde cada registro é composto por três partes:

- Um rótulo iniciado pelo caractere ">" e seguido pelo nome e origem da sequência;
- A sequência de nucleotídeos, composta pelos caracteres A, T, C, G ou N em caso de *gaps*;
- Um caractere "*" opcional para indicar o final da sequência.

Um exemplo de arquivo *FASTA* com duas sequências é ilustrado na Figura 1. Repare que essas sequências apresentam bases ainda não identificadas, que são representadas pela letra N.

Figura 1 – Exemplo de arquivo *FASTA* com duas sequências de nucleotídeos

exemplo-formato-fasta.txt

```
>Contig1_Corynebacterium
TCATCATGATAAGCTCAGAAACACCGTTGCCTAGATACACGTCGTCTACGTCGAAGGCAG
GGAAGCNNNNNNNNNNNNNNNNNNNNNNNNCGATGGCGCGCCGGGCAGGAATGATGCCTT
TTGACGTGGAATAGCCTTGCGACGTAGAAAGCGCCGCGATCATGTCACG
>Contig2_Corynebacterium
CCCTTTGTCGCCCGATGTGAAGGAAAANNNNNNNNNNNNNNNNNNNNTTTTTCTTCTTGT
GACCCTTCTTACTAGAGTTTGGAAAGTAAAATTATCGCCACACCCTTTGCACTAAGCATG
ACTGAGTAATGGTCGCATGGTTAAACCTGCCAATGAGT
```

Fonte: O Autor (2017)

Na análise, foi definido como parâmetro de entrada no *GapBlaster* o argumento "Tamanho Mínimo da Região Flanqueadora = 11" e foi selecionado o *Blast+* como alinhador padrão. É importante ressaltar que as simulações foram executadas na versão 1.1.1 do *GapBlaster* e que os resultados foram avaliados manualmente. Dessa forma, somente foram aceitos os alinhamentos cujas regiões flanqueadoras possuíssem alta semelhança com o genoma de referência.

1.4.4 Comparação do Fechamento de Gaps

Foram realizadas comparações com as ferramentas *FGAP* e *GapFiller* em seus parâmetros padrões para avaliar a performance obtida pelo *GapBlaster*. O programa *FGAP* aborda o problema de fechamento de *gaps* de maneira bastante similar ao *GapBlaster*, utilizando *contigs* para fechar os *gaps* (PIRO et al., 2014). Por outro lado, o programa *GapFiller* aborda o mesmo problema através do mapeamento iterativo de leituras pareadas contra os *scaffolds*, sendo que desta forma ele pode inclusive prever o tamanho dos *gaps* e verificar a corretude dos *contigs* (BOETZER; PIROVANO, 2012).

Nesta análise foram utilizados os dados descritos na subseção 1.4.2. Contudo, o genoma da bactéria *Corynebacterium pseudotuberculosis* foi analisado somente no *GapBlaster* e no *FGAP*, já que este organismo foi sequenciado através de biblioteca de fragmentos e o *GapFiller* requer biblioteca de pares interligados como entrada.

Adicionalmente, foram realizados experimentos onde os resultados gerados pelo *FGAP* e pelo *GapFiller* foram utilizados como entrada no *GapBlaster*, com o propósito de verificar se haveria uma melhora significativa nas montagens ao utilizar o *GapBlaster* em conjunto com outras ferramentas.

É importante ressaltar que os resultados obtidos pelo *GapBlaster* nos experimentos realizados foram influenciados pela visão de especialistas, que selecionaram os *gaps* que seriam fechados durante a análise através da interface gráfica do *GapBlaster*. Os especialistas em questão eram alunos de doutorado do Programa de Pós-Graduação em Genética e Biologia Molecular na ocasião e a seleção dos alinhamentos foi feita com base no grau de similaridade entre o *scaffold* e os *contigs*. Ou seja, alinhamentos muito discrepantes foram descartados.

1.4.5 Avaliação dos Resultados

A qualidade dos resultados foi avaliada através da contagem do número de *gaps* e do total de Ns dos *scaffolds* originais e também dos *scaffolds* obtidos nos testes com o *GapBlaster*, *FGAP* e *GapFiller*. Esta contagem foi feita por um *script* desenvolvido na linguagem de programação *perl*, o qual pode ser encontrado no endereço <https://sourceforge.net/projects/gapblaster/files/scripts/>.

A confirmação de que os *gaps* foram fechados corretamente foi feita através do *script* de validação disponibilizado pelo projeto GAGE no endereço <http://gage.cbcb.umd.edu/results/gage-paper-validation.tar.gz>. Os genomas de referência e os arquivos *FASTA* contendo os *scaffolds* originais ou os *scaffolds* com os *gaps* fechados foram usados como entrada no *script*.

Informações sobre esses genomas de referência são dadas na Tabela 2. As linhas dessa tabela listam respectivamente a quantidade total de pares de bases do genoma de referência, a porcentagem de bases G e C presentes no genoma, a quantidade de cromossomos, a quantidade de plasmídeos, o código de acesso do genoma no banco de dados *Genbank* e o total de repetições

de 150 e 250 pares de bases.

Tabela 2 – Dados dos genomas de referência usados para validar os *gaps* fechados

Organismo	***C. pseudotuberculosis 262***	***S. aureus A–S391_USA300***	***R. sphaeroides 2.4.1***
Tamanho do Genoma	2325749	2872916	4628173
Conteúdo GC	52,17	32,76	68,77
Nº de Cromossomos	1	1	2
Nº de Plasmídeos	0	0	5
Genbank	CP012022.1	CP007690.1	GCA_000273405.1
Repetições de 150 pb	8007	10709	38073
Repetições de 250 pb	6612	9460	35353

Fonte: Sá et al. (2016)

1.5 Estrutura do Trabalho

Além desta Introdução, foram elaborados mais 4 capítulos neste trabalho, onde são dissertados os tópicos listados abaixo a fim de fornecer um melhor entendimento sobre o projeto de construção de software que foi executado. Esses capítulos são:

- **Capítulo 2 - Sequenciamento e Montagem de Genomas**

Neste capítulo são explicadas as principais metodologias existentes para a realização das etapas de sequenciamento e montagem de genomas, que ocorrem antes da etapa de curadoria.

- **Capítulo 3 - Implementação**

Neste capítulo é exposto o algoritmo utilizado no *GapBlaster*, além da solução desenvolvida para auxiliar pesquisadores na etapa de fechamento de *gaps* e as considerações a respeito de sua utilização.

- **Capítulo 4 - Resultados e Discussão**

Neste capítulo é relatado o desempenho do *GapBlaster* ao utilizar-se dados reais de montagens, assim como a descrição dos resultados obtidos e a argumentação com base nesses resultados.

- **Capítulo 5 - Considerações Finais**

Neste último capítulo é apresentada a conclusão acerca do trabalho realizado e são propostas sugestões para trabalhos futuros.

2 SEQUENCIAMENTO E MONTAGEM DE GENOMAS

2.1 Uma Breve História do Sequenciamento de DNA

A genômica é uma área de pesquisa relativamente nova, que engloba todo e qualquer estudo do genoma. Ela teve seu início por volta dos anos 50, sendo que uma das descobertas mais importantes dessa época ocorreu quando *James Watson* e *Francis Crick* descobriram a estrutura helicoidal da molécula de DNA, retratada na Figura 2 (WATSON; CRICK, 1953).

Figura 2 – Estrutura da molécula de DNA

Fonte: O Autor (2017)

Posteriormente, na década de 60, o Dr. *Frederick Sanger* publicou um método para sequenciar RNA (SANGER; BROWNLEE; BARRELL, 1965). Isto o motivou a pesquisar junto com sua equipe um método para sequenciar DNA e assim surgiu a publicação do renomado Método de *Sanger*, onde é descrita uma técnica que essencialmente faz com que a DNA polimerase se incorpore aos nucleotídeos através uma pequena modificação química (SANGER; NICKLEN; COULSON, 1977).

Apesar dos esforços para automatizar algumas etapas do processo de sequenciamento pelo método de *Sanger*, a técnica era altamente laboral e não se mostrou escalável (MARDIS, 2013). Assim sendo, uma nova metodologia de sequenciamento era necessária para gerar maior vazão de dados. Isto motivou a criação dos sequenciadores de nova geração, os quais serão descritos na próxima seção.

2.2 Visão Geral dos Sequenciadores de Nova Geração

De acordo com Reis-Filho (2009), a maior diferença entre os sequenciadores de *Sanger* e os sequenciadores de nova geração (NGS) é que estes podem processar milhões de leituras em paralelo. Reis-Filho (2009) nota que essa alta vazão implica que em apenas uma ou duas rodadas de sequenciamento é possível completar um experimento.

Outra característica importante é que as amplificações das plataformas NGS são produzidas *in vitro* ao invés de serem clonadas em vetores, como ocorria no sequenciamento capilar (AIRD et al., 2011). Dessa forma, os problemas de contaminação das amostras são evitados. Apesar disso, cada plataforma NGS apresenta suas falhas específicas, por exemplo: inserção, deleção e substituição de bases (WOJCIESZEK et al., 2014).

Segundo Mardis (2008), o processo para produzir bibliotecas para os sequenciadores de nova geração é bem mais prático que no Método de *Sanger*, pois os fragmentos de DNA são preparados para sequenciamento simplesmente através da ligação de adaptadores em ambas extremidades de cada fragmento de DNA. Mardis (2008) observa que o mais importante é que apenas alguns microgramas de DNA são necessários para produzir uma biblioteca de fragmentos.

Contudo, as plataformas NGS também tem a habilidade de sequenciar os pares interligados de um dado fragmento, usando um processo sutilmente modificado para criação da biblioteca (MARDIS, 2008). Esta abordagem pode ser utilizada se um genoma for montado através de uma estratégia *de novo* a partir de dados NGS, por exemplo.

Em relação ao tamanho das leituras geradas, os sequenciadores NGS produzem leituras mais curtas, em média 35–250 pb dependendo da plataforma, enquanto sequenciadores capilares geram leituras entre 650–800 pb (MARDIS, 2008). Esta característica faz com que leituras produzidas pelas plataformas NGS necessitem de uma cobertura em torno de 25–30 repetições de cada amostra para que toda a informação genética possa ser capturada (MARDIS, 2008).

Por último, as plataformas NGS também acabam gerando economia para os laboratórios de pesquisa. De acordo com Reis-Filho (2009), o paralelismo dos sequenciadores NGS é tão eficiente que é possível sequenciar 30Gb de sequencias do DNA humano em uma única rodada de experimentos por menos de US$20.000, enquanto o Consórcio Internacional de Sequenciamento do Genoma Humano gerou apenas 3Gb ao custo de US$3 bilhões e levou 13 anos para tal. Esta economia provida pelas plataformas NGS acabou tornando o sequenciamento de genomas acessível para pequenos laboratórios.

Após o sequenciamento do genoma ter sido realizado, é necessário que as leituras sejam organizadas a fim de se obter a informação genética original. Esta etapa é conhecida como montagem e será vista com maiores detalhes na próxima seção.

2.3 Montagem de Genomas

Ainda que a tecnologia NGS tenha trazido avanços significativos para a ciência, ela também trouxe novos problemas para serem resolvidos. Talvez o mais notável de todos seja o problema para montar os dados NGS, pois as características das leituras eram diferentes das até então geradas pelos sequenciadores capilares. Isso significa que os montadores existentes na época não seriam capazes de manipular corretamente o tamanho reduzido das leituras NGS nem

a alta redundância de dados gerados.

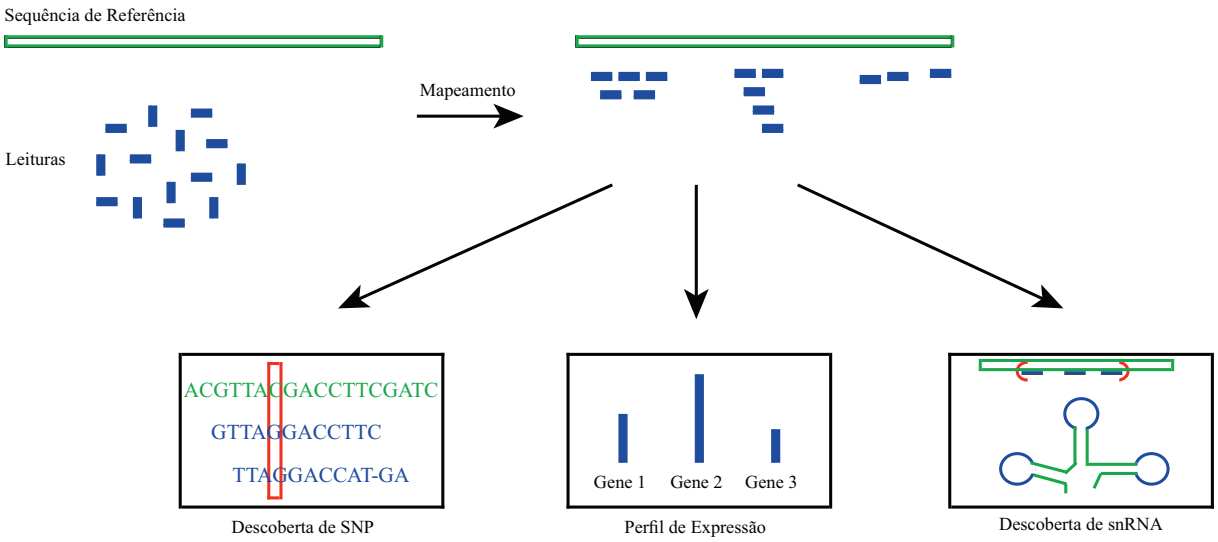
Por estes motivos, foi necessário criar novos montadores que pudessem gerenciar eficientemente o grande volume de dados gerados pelos sequenciadores NGS. Nas próximas subseções serão explicadas em maiores detalhes duas estratégias de montagem que são utilizadas para tal finalidade.

2.3.1 Mapeamento e Montagem Comparativa

Atualmente, devido a grande quantidade de genomas publicados em bancos de dados públicos, frequentemente ocorre em muitos projetos de sequenciamento o fato de já existir o genoma finalizado de um organismo similar ao que se quer montar. Isto pode simplificar bastante o processo de montagem do genoma, visto que para executar a tarefa basta apenas alinhar as sequências contra o genoma de referência para descobrir o posicionamento das mesmas e realizar a montagem *de novo* (descrita na próxima subseção) somente das leituras que não foram mapeadas (NAGARAJAN; POP, 2010).

Esta estratégia tem sido muito utilizada em projetos de ressequenciamento para montar bactérias que são geneticamente relacionadas e também em outras aplicações, que vão desde a descoberta de RNA não codificante até a caracterização do padrão de metilação (NAGARAJAN; POP, 2010). Na Figura 3, é possível ver como se dá o funcionamento do processo de mapeamento e montagem comparativa na descoberta de polimorfismo de base única, na descoberta do perfil de expressão gênica e na descoberta de snRNA. Nesta ilustração, as leituras são mapeadas contra um genoma de referência, de forma a determinar o posicionamento das leituras em relação ao genoma de referência.

Figura 3 – Mapeamento de leituras e aplicações



Fonte: Nagarajan e Pop (2010)

Com base nesses mapeamentos, é possível descobrir, por exemplo, quais nucleotídeos

mudam de um organismo para o outro. Esse processo é conhecido como descoberta de polimorfismo de base única (SNP na abreviação em inglês). No canto inferior esquerdo da Figura 3 é ilustrado esse processo. Repare que as leituras mapeadas, que estão em cor azul, ficam alinhadas perfeitamente em relação à sequência de referência, que está na cor verde. Com base nestes alinhamentos, é possível identificar que apenas uma base mudou nesta região do genoma que foi sequenciado.

Com base nos mapeamentos realizados contra um genoma de referência, também é possível descobrir quais informações genéticas são hereditárias no genoma que está sendo montado. Esta descoberta é feita através do perfil de expressão gênica, que identifica com base nas sequências mapeadas quais são os genes existentes no genoma de referência que também existem no genoma que está sendo montado (ver centro inferior da Figura 3).

Uma terceira aplicação do mapeamento e montagem comparativa consiste na descoberta de snRNA, que são envolvidos na união e outras reações de processamento de RNA, conforme exibe o canto inferior direito da Figura 3.

Ainda existe uma segunda abordagem de montagem que é utilizada quando não existe um genoma de referência, a qual é denominada montagem *de novo*. Os detalhes do funcionamento da montagem *de novo* serão explicados na próxima subseção.

2.3.2 Montagem *de novo*

A montagem *de novo* tem como objetivo reconstruir o genoma de determinado organismo somente a partir das leituras que foram sequenciadas. Segundo Nagarajan e Pop (2010), os algoritmos de montagem *de novo* se baseiam no pressuposto que duas leituras sobrepostas em grande parte de suas extensões provavelmente representam segmentos vizinhos de um genoma.

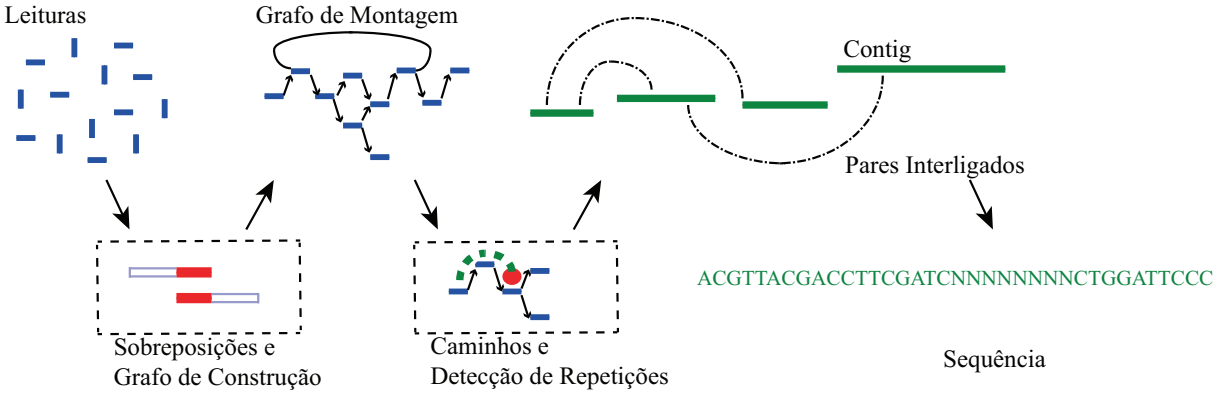
Porém, conforme Nagarajan e Pop (2010) bem notam, este pressuposto é inválido quando as sequências que se sobrepõem são parte de uma região repetitiva do genoma. Desta forma, os tamanhos das leituras dos sequenciadores NGS implicam que mesmo pequenas regiões de repetição podem introduzir ambiguidades no processo de montagem. Logo, reconhecer tais regiões é uma parte importante no processo de montagem de genomas.

Uma visão geral do processo de montagem *de novo* é ilustrada na Figura 4. Nesta ilustração, o processo de montagem *de novo* é iniciado a partir da obtenção das leituras curtas sequenciadas por uma plataforma NGS. Essas leituras são então alinhadas umas com as outras para verificar se ocorre a sobreposição do sufixo de uma leitura com o prefixo de outra leitura. Desta forma, é gerado um grafo de montagem que representa todas as sobreposições de leituras identificadas.

Em seguida, um algoritmo percorre esse grafo de montagem a fim de detectar agregações de leituras. Os caminhos identificados são denominados *contigs*. Porém, além de identificar os *contigs*, essa caminhada no grafo de montagem também tenta identificar regiões repetitivas

do genoma. Essas regiões repetitivas devem ser evitadas pelo montador, pois geralmente são complexas demais para serem montadas a partir de leituras curtas.

Figura 4 – Visão geral da montagem *de novo*



Fonte: Nagarajan e Pop (2010)

Por último, o montador ordena os *contigs*. Essa ordenação é denominada de *scaffolds*. Nos *scaffolds* são inseridos *gaps* quando as leituras presentes no final de um *contig* se sobrepõem com leituras de dois ou mais *contigs*, já que este é um indicativo de região repetitiva. Os tamanhos dos *gaps* podem ser calculados com base no tamanho de inserção das leituras curtas presentes nos *contigs*.

Desta forma, a saída gerada pelos montadores *de novo* é frequentemente fragmentada e cheia de *gaps*. Quando isso ocorre, é necessário realizar mais uma etapa denominada curadoria para fechar as lacunas remanescentes. O programa desenvolvido neste trabalho visa justamente auxiliar nessa etapa de curadoria. Os detalhes de sua implementação são descritos no capítulo seguinte.

3 IMPLEMENTAÇÃO

3.1 Algoritmo do GapBlaster

Em síntese, o algoritmo do *GapBlaster* realiza sete etapas para fechar *gaps*. Primeiramente, os arquivos com os *contigs* obtidos da montagem do genoma são concatenados em um arquivo temporário. Após a concatenação, os *contigs* são alinhados contra os *scaffolds* usando o alinhador de preferência do usuário. Os resultados dos alinhamentos são então convertidos para um formato próprio do *GapBlaster* e os *contigs* são ordenados de acordo com a posição mapeada nos *scaffolds*.

Em seguida, o programa concatena os alinhamentos pertencentes ao mesmo *contig* que flanqueiam regiões com *gap*. Todos os alinhamentos identificados que fecham *gaps* são apresentados ao usuário para inspeção através da interface gráfica do *GapBlaster* e o usuário pode escolher quais alinhamentos serão usados para fechar *gaps*. Por último, os *gaps* são fechados com base nas escolhas feitas pelo usuário e é gerado um relatório com as alterações efetuadas.

Todos os passos mencionados acima estão listados como funções no Algoritmo 1. Cada uma das funções delineadas no Algoritmo 1 será explicada em maiores detalhes nas próximas subseções.

Algoritmo 1: GAPBLASTER

Entrada: $Scaffolds$, $ArquivosComContigs[1...*]$
Saída: *Scaffolds* curados e Relatório das alterações feitas

1 **início**
2 $\quad ContigsConcatenados = \text{CONCATENAR}(ArquivosComContigs[1...*])$
3 $\quad Alinhamentos = \text{ALINHAR}(Scaffolds, ContigsConcatenados)$
4 $\quad AlinhamentosConvertidos = \text{CONVERTER}(Alinhamentos)$
5 $\quad AlinhamentosOrdenados = \text{ORDENAR}(AlinhamentosConvertidos)$
6 $\quad AlinhamentosConcatenados = \text{CONCATENAR}(AlinhamentosOrdenados)$
7 $\quad AlinhamentosEscolhidos = \text{EXIBIR}(AlinhamentosConcatenados)$
8 $\quad \text{FECHARGAPS}(Scaffolds, AlinhamentosEscolhidos, ScaffoldsCurados, Relatorio)$
9 **fim**
10 **retorna** $ScaffoldsCurados, Relatorio$

3.1.1 Concatenação dos Arquivos de *Contigs*

A concatenação dos diferentes arquivos que contém os *contigs* é feita através de um arquivo de texto criado temporariamente no sistema de arquivos do usuário. Este arquivo temporário é nomeado com o prefixo `"gapblaster_"`, uma numeração automática gerada pela classe *File* do *Java* e o sufixo `"_mergedContigs"`. Por exemplo, em um experimento realizado

foi gerado o arquivo "`gapblaster_6249295881001741855_mergedContigs`" no diretório de arquivos temporários.

Esta concatenação dos arquivos que contém os *contigs* é feita porque o *Legacy Blast* e o *Blast+* aceitam como parâmetro apenas um único arquivo *FASTA* com os *contigs* que serão alinhados. Caso contrário, se forem enviados dois ou mais arquivos, estes programas não reconhecem os parâmetros enviados e consequentemente não executam os alinhamentos solicitados. Desta forma, a melhor solução encontrada foi concatenar esses arquivos.

3.1.2 Alinhamento dos *Contigs* Contra os *Scaffolds*

Os alinhamentos dos *contigs* contra os *scaffolds* são feitos a partir de uma chamada de sistema ao alinhador predefinido pelo usuário. Por esta razão, é necessário que o alinhador utilizado esteja na variável de ambiente *path* do sistema operacional. Caso contrário, o *GapBlaster* não conseguirá chamar o alinhador e consequentemente exibirá uma mensagem de erro e terminará a execução.

Na versão atual, o *GapBlaster* é compatível com os alinhadores *Legacy Blast*, *Blast+* e *MUMmer*. Porém, mais alinhadores podem ser adicionados através da conversão da saída gerada pelo alinhador desejado para o formato utilizado pelo *GapBlaster*, conforme descrito na próxima subseção.

3.1.3 Conversão dos Alinhamentos

Os alinhamentos descritos na seção anterior precisam ser convertidos para um formato que é utilizado pelo *GapBlaster* nas demais etapas de processamento. Desta forma, a inclusão de novos alinhadores é simplificada, pois se torna necessário apenas implementar um novo conversor para o alinhador que será adicionado ao invés de alterar todo o código-fonte do programa.

Os alinhamentos convertidos para o formato padrão do *GapBlaster* devem conter as seguintes informações, separadas por linhas:

- Sequência do *scaffold*;
- Linha do meio, que indica se as bases alinharam corretamente;
- Sequência do *contig*;
- Nome do *contig*;
- Nome do *scaffold*;
- Tamanho do alinhamento;
- Início do alinhamento no *contig*;

- Final do alinhamento no *contig*;
- Início do alinhamento no *scaffold*;
- Final do alinhamento no *scaffold*;
- Booleano indicando se o alinhamento está na fita reversa (verdadeiro ou falso em inglês);
- Linha em branco para separar do próximo alinhamento.

A Figura 5 exibe, a título de exemplo, um alinhamento que foi convertido para o formato utilizado pelo *GapBlaster* e que, portanto, contém todas as informações descritas na lista acima.

Figura 5 – Exemplo de arquivo de texto com um alinhamento convertido para o formato utilizado pelo *GapBlaster*

exemplo-formato-gapblaster.txt

```
TTTCTTCTCCTACGGGTACTGAGATGTTTCACTTCCC
||||||||||||||||||||||||||||||||||||
TTTCTTCTCCTACGGGTACTGAGATGTTTCACTTCCC
NÓ_3671_tamanho_96_cob_423.854_ID_7341
NÓ_3710_tamanho_89_cob_30.2059_ID_7419
37
1
37
1
37
true
```

Fonte: O Autor (2017)

3.1.4 Ordenação dos Alinhamentos

Os alinhamentos são ordenados em ordem crescente e com base em diversos atributos, a fim de resolver os casos em que os atributos comparados são similares. Desta forma, a ordenação é feita a partir dos atributos seguindo esta ordem: nome do *scaffold*, nome do *contig*, início do alinhamento no *scaffold* e início do alinhamento no *contig*.

3.1.5 Concatenação dos Alinhamentos

Frequentemente, os alinhadores fragmentam o mapeamento de um *contig* contra um *scaffold* em dois ou mais alinhamentos com o objetivo de aumentar o grau de compatibilidade entre as sequências alinhadas. Contudo, estes alinhamentos fragmentados podem pertencer à regiões flanqueadoras de *gaps*. Logo, seria interessante uni-los com o propósito de fechar *gaps*. Por este motivo, foi implementado no *GapBlaster* um algoritmo que concatena os alinhamentos

fragmentados, desde que eles pertençam ao mesmo *contig* e *scaffold*. Além disso, este algoritmo leva em conta também se as sequências de ambos alinhamentos estão na fita reversa ou não e é feito o ajuste em um dos alinhamentos em caso negativo.

3.1.6 Exibição dos Alinhamentos

Os alinhamentos identificados que fecham *gaps* são apresentados ao usuário por meio da interface gráfica do programa, para que o usuário possa então escolher quais alinhamentos deseja utilizar para fechar *gaps*. A decisão de manter a seleção dos alinhamentos de forma manual se deu a partir da hipótese que seria possível a utilização de parâmetros de alinhamentos menos rígidos se o usuário pudesse validá-los posteriormente através de inspeção visual.

3.1.7 Fechamento dos *gaps*

Por último, os *gaps* são fechados a partir dos alinhamentos selecionados pelo usuário. Contudo, é importante ressaltar que diferentes alinhamentos apresentados ao usuário podem fechar o mesmo *gap* e, portanto, a quantidade de *gaps* fechados ao final da execução do *GapBlaster* pode diferir da quantidade de seleções feitas pelo usuário. Ou seja, podem ser fechados menos *gaps* que o esperado.

Após o processo de fechamento de *gaps* ser concluído, é apresentada uma mensagem ao usuário informando a quantidade de *gaps* e Ns fechados e o programa retorna à tela principal. São gerados dois arquivos de saída pelo *GapBlaster*. O primeiro arquivo possui os *scaffolds* curados pelo *GapBlaster* e tem o mesmo nome do arquivo com os *scaffolds* originais acrescido da extensão "_gapblasted.fasta". O segundo arquivo criado pelo *GapBlaster* é um relatório com as alterações feitas no *scaffold*. Este relatório apresenta o mesmo formato especificado na subseção 3.1.3 e é salvo em um arquivo com o nome "gapblaster.log".

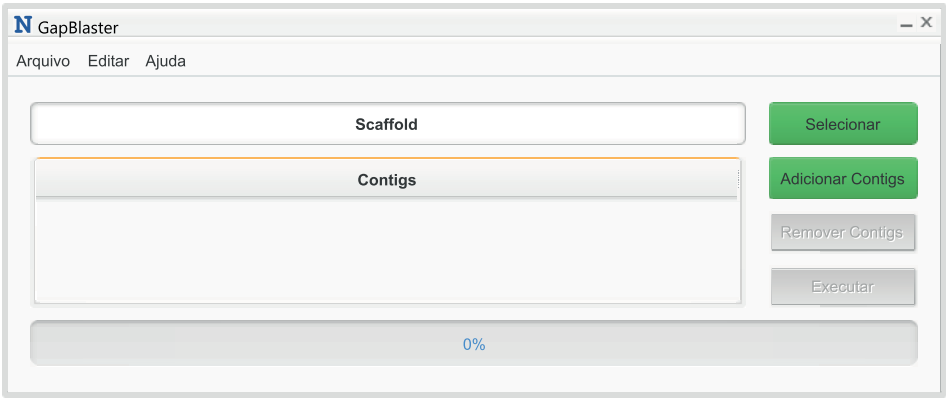
3.2 Interface Gráfica

Na tela principal do GapBlaster, o usuário pode selecionar os arquivos *FASTA* que contém os *scaffolds* e os *contigs*, como exibe a Figura 6. A seleção do arquivo *FASTA* que contém os *scaffolds* é feita ao clicar no botão selecionar. Ao clicar neste botão, é exibida uma janela que permite realizar a seleção do arquivo *FASTA* desejado. Após feita a escolha, a palavra *Scaffolds* é substituída pelo caminho do arquivo selecionado na caixa de texto à esquerda.

De forma similar, o usuário pode adicionar arquivos que contém *contigs* ao clicar no botão Adicionar *Contigs* na tela principal do *GapBlaster* (ver Figura 6). Os caminhos dos arquivos de *contigs* adicionados são exibidos na lista à esquerda e ao selecionar um dos arquivos dessa lista o botão Remover *Contigs* fica habilitado para permitir a exclusão do arquivo selecionado.

Outra forma do usuário selecionar os arquivos de *scaffolds* e *contigs* é por meio do menu arquivo, que fica no topo da tela principal. No menu arquivo, também é possível finalizar a execução do programa ao clicar na opção Sair. Caso o usuário queira acessar o manual de instruções, é possível abri-lo por meio do menu Ajuda. A versão do programa é exibida ao escolher a opção Sobre do menu Ajuda.

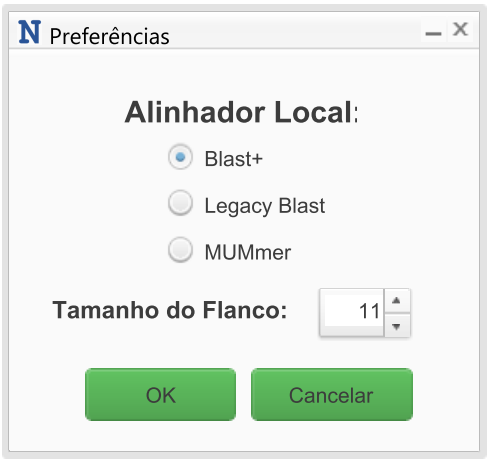
Figura 6 – Tela principal do programa, onde o usuário pode selecionar os arquivos contendo os *scaffolds* e *contigs* e tem acesso ao menu de opções



Fonte: O Autor (2017)

Por meio da opção Preferências do menu Editar, o programa exibe a tela de configuração retratada na Figura 7. Nesta janela, é possível que o usuário escolha o alinhador padrão, entre *Legacy Blast*, *Blast+* ou *MUMmer*, bem como o tamanho mínimo da região flanqueadora.

Figura 7 – Tela de configuração, onde o usuário pode definir suas preferências

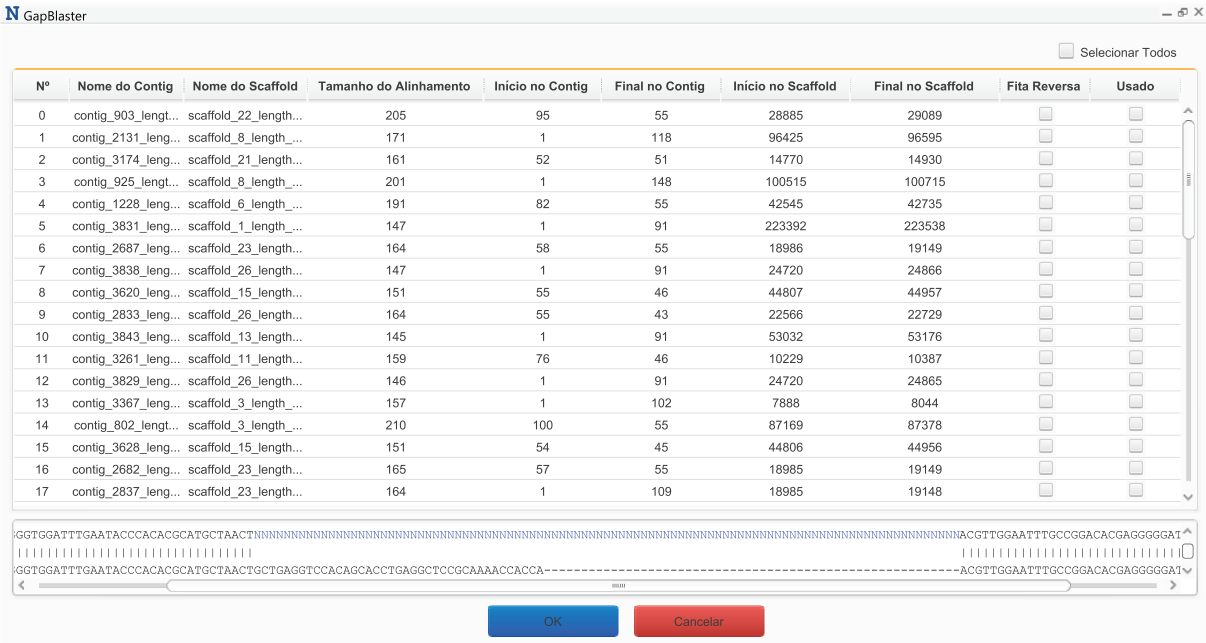


Fonte: O Autor (2017)

Após serem selecionados os arquivos contendo os *scaffolds* e *contigs*, o botão executar é habilitado na tela principal e o usuário pode então clicá-lo para que o *GapBlaster* inicie o processamento dos arquivos. Quando o processamento é concluído, outra tela exibe os resultados dos alinhamentos (ver Figura 8) e o usuário pode então realizar curadoria manual e selecionar os alinhamentos que melhor fecham os *gaps*.

Para auxiliar o usuário na escolha dos melhores alinhamentos, a tabela de resultados retratada na Figura 8 contém diversas informações sobre os alinhamentos, como por exemplo: o tamanho dos alinhamentos, a posição inicial e final do alinhamento no *scaffold* e no *contig* e uma caixa de seleção que informa se o alinhamento está na fita reversa. Esta caixa de seleção fica desabilitada para o usuário.

Figura 8 – Esta janela mostra todos os resultados de alinhamentos que podem fechar *gaps* e permite ao usuário escolher quais serão utilizados para tal finalidade



Fonte: O Autor (2017)

O usuário pode ordenar em ordem crescente os resultados exibidos na tabela da Figura 8 ao clicar no nome de uma das colunas. Caso o nome da coluna seja clicado novamente, a ordenação fica em ordem decrescente.

A curadoria manual dos alinhamentos pode ser feita ao clicar em uma das linhas da tabela de resultados. Com esta ação, a parte inferior da tela de resultados passa a exibir o alinhamento selecionado. Desta forma, o usuário pode realizar a curadoria com maior segurança ao encontrar *contigs* que se alinham às regiões flanqueadores do *gap* e fecham o *gap* completamente, conforme exibido na parte inferior da Figura 8.

Um alinhamento pode ser selecionado para fechar um *gap* ao clicar na caixa de seleção Usado. Alternativamente, todos os alinhamentos podem ser utilizados para fechar *gaps* ao clicar na caixa de seleção Selecionar Todos.

Após o usuário ter selecionado os *gaps* que serão fechados, ele pode clicar no botão *OK* e o *GapBlaster* fará as alterações solicitadas e gerará um relatório com as alterações que foram feitas nos *scaffolds*. Por último, o programa finaliza sua rotina retornando à tela principal.

4 RESULTADOS E DISCUSSÃO

4.1 Dados de Teste

O desempenho do *GapBlaster* foi avaliado através de experimentos de fechamento de *gaps* em dados reais de montagens. Os dados utilizados são oriundos da montagem do genoma da bactéria *C. pseudotuberculosis* através do montador *SPADES* e também das montagens produzidas por diversos montadores das bactérias *S. aureus* e *R. sphaeroides*. O número de bases ausentes e de *scaffolds* resultantes dessas montagens são exibidos na Tabela 3.

Tabela 3 – Informações de montagem dos genomas de referência

Organismo	Montador	Bases com N	*Scaffolds*
Corynebacterium pseudotuberculosis 262			
	SPADES	2893857	4611
Staphylococcus aureus A–S391_USA300			
	ABySS	3893185	5012
	ABySS2	3821622	125
	Allpaths-LG	2880676	19
	Bambus2	2862930	17
	MSR-CA	2872905	17
	SGA	3128388	546
	SOAPdenovo	2924135	175
	Velvet	2877995	173
Rhodobacter sphaeroides 2.4.1			
	ABySS	5160167	2714
	ABySS2	5331930	480
	Allpaths-LG	4609785	38
	Bambus2	4428612	92
	CABOG	4259679	130
	MSR-CA	4498559	44
	SGA	5614693	2096
	SOAPdenovo	4627058	312
	Velvet	4615068	382

Fonte: Sá et al. (2016)

4.2 Análise de Fechamento de Gaps

Os resultados alcançados pelo *GapBlaster* foram validados através da comparação com os resultados obtidos pelos programas *FGAP* e *GapFiller*. Porém, como o *GapFiller* somente aceita como entrada bibliotecas de pares interligados e o genoma da bactéria *C. pseudotuberculosis* foi sequenciado através de biblioteca de fragmentos, este genoma não pôde ser testado no *GapFiller*.

Após a execução da curadoria com o *GapBlaster*, o total de *gaps* resultantes da montagem da bactéria *C. pseudotuberculosis* foi reduzido de 24 para 11, enquanto que com o programa *FGAP* a redução foi de 24 para 5 *gaps*. Estes resultados são exibidos no topo da Tabela 4.

Tabela 4 – Resultados do fechamento de *gaps*

Organismo	Montador	#Gaps	#N	#Gaps GB	#N GB	#Gaps *FGAP*	#N *FGAP*	#Gaps GF	#N GF
C. pseudotuberculosis									
	SPADES	24	1794	11	931	5	360	–	–
S. aureus									
	AbySS	66	55882	55	47614	45	51127	69	56355
	AbySS2	33	9391	27	7780	17	4850	35	10003
	Allpaths-LG	23	9875	20	9446	15	8755	40	10472
	Bambus2	95	29201	93	29159	80	27459	98	30771
	MSR-CA	81	10353	72	7868	47	7861	80	11651
	SGA	654	300607	642	292067	634	298252	654	312284
	SOAPdenovo	9	4857	8	4837	7	4708	9	5010
	Velvet	128	17688	124	17473	94	15406	127	19863
R. sphaeroides									
	AbySS	261	114525	261	114525	256	113886	306	118298
	AbySS2	235	62570	233	62128	228	60323	290	68052
	Allpaths-LG	90	21329	87	20733	82	19500	164	24001
	Bambus2	85	57041	83	56402	80	55990	84	56930
	CABOG	193	21547	192	20892	190	21065	191	25011
	MSR-CA	356	32628	349	26189	347	31174	336	37494
	SGA	938	1145600	938	1145600	930	1144955	930	1159235
	SOAPdenovo	38	10461	37	9601	37	10097	38	11176
	Velvet	427	86815	424	86785	404	86063	415	94150

Fonte: Sá et al. (2016)

Os dados de montagem das bactérias *S. aureus* e *R. sphaeroides*, obtidos a partir do projeto GAGE, foram gerados por diversos montadores e seus *contigs* e *scaffolds* foram submetidos para os programas *GapBlaster*, *FGAP* e *GapFiller*. As análises com o *GapBlaster* revelaram reduções nas quantidades de *gaps* e Ns em todas as montagens da bactéria *S. aureus*. Contudo, ao serem utilizados como entrada no *GapBlaster* os dados referentes à bactéria *R. sphaeroides*, não houve redução no número de *gaps* somente nos dados gerados pelos montadores *AbySS* e SGA, conforme demonstra a Tabela 4.

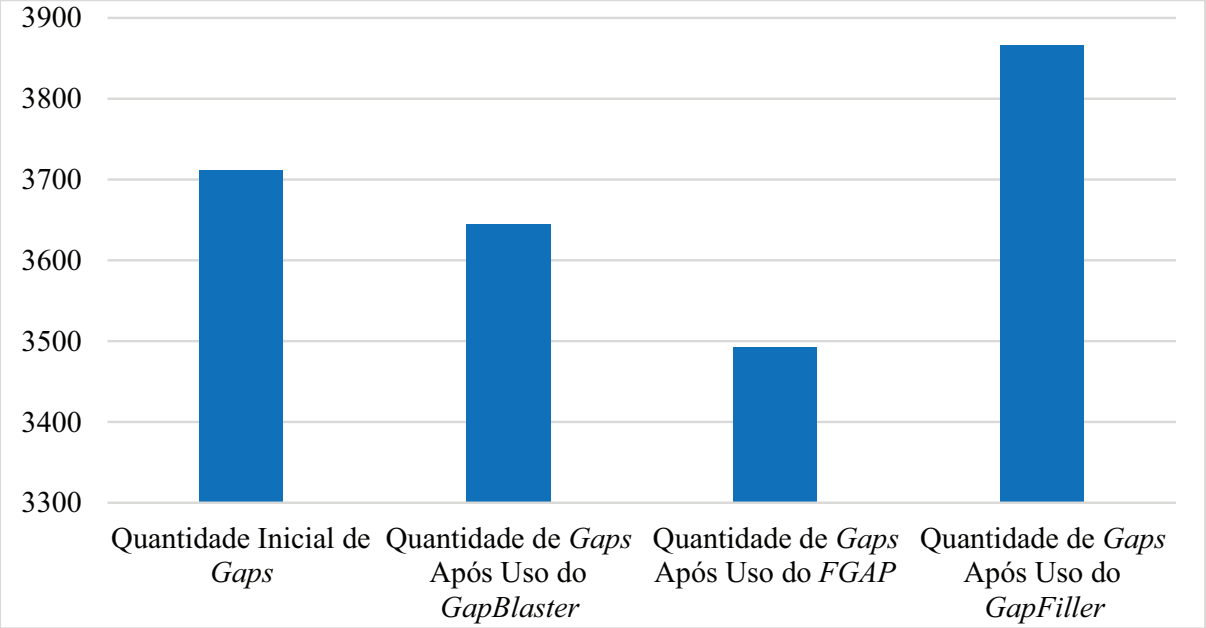
Comparando o desempenho do *GapBlaster* com os resultados obtidos pelo programa *GapFiller* (ver últimas colunas da Tabela 4), pode-se constatar que o *GapFiller* aumentou ou manteve a mesma quantidade de *gaps* em cerca de 59% das montagens de ambos organismos. Nos demais casos, a maioria absoluta das montagens teve a quantidade de bases desconhecidas, que são representadas por Ns, aumentadas. Estas ocorrências são causadas pelo tamanho de inserção utilizado para alinhar as leituras pareadas contra as sequências de referência. Desta forma, é possível afirmar que a performance do *GapBlaster* foi superior a obtida pelo *GapFiller* na curadoria de genomas, pois o *GapBlaster* não somente fechou mais *gaps* como também reduziu a quantidade de bases desconhecidas em quase todas as montagens do projeto GAGE.

Prosseguindo com a análise, foi constatado que o programa *FGAP* fechou mais *gaps* que o *GapBlaster* em todas as montagens obtidas do projeto GAGE. Entretanto, o *GapBlaster* obteve uma redução maior na quantidade de bases desconhecidas nos dados gerados pelos montadores

ABySS e SGA para a bactéria *S. aureus* e também nos dados gerados pelos montadores *CABOG*, MSR-CA e *SOAPdenovo* para a bactéria *R. sphaeroides* (ver Tabela 4).

Os resultados obtidos na análise de fechamento de *gaps* podem ser resumidos no Gráfico 2, que foi elaborado através da contagem do total de *gaps* antes e depois da curadoria nos dados de montagem das bactérias *S. aureus* e *R. sphaeroides*. Neste gráfico, é perceptível que o *FGAP* foi o programa que mais fechou *gaps*, seguido pelo *GapBlaster*, enquanto o *GapFiller* aumentou a quantidade de *gaps*.

Gráfico 2 – Síntese dos resultados de fechamento de *gaps*



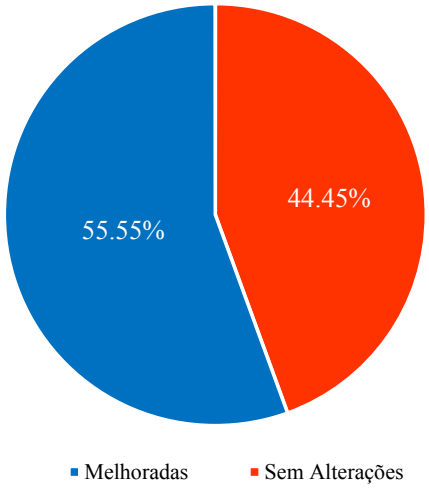
Fonte: Dados obtidos de Sá et al. (2016)

4.3 Análise de uso do GapBlaster em Conjunto com Outros Programas de Curadoria

Embora o *FGAP* tenha mostrado melhores resultados no fechamento de *gaps* para os dados dos três organismos, foi realizada uma subsequente análise de fechamento de *gaps*, cujo objetivo era determinar se o *GapBlaster* poderia melhorar os resultados produzidos por outros programas de curadoria de genomas.

Para atingir este objetivo, primeiramente foram utilizados como entrada no *GapBlaster* os *scaffolds* modificados pelo *FGAP* em conjunto com os *contigs* originais de cada montagem. Os resultados retratados no Gráfico 3 mostram que o *GapBlaster* aprimorou 55.55% de todas as montagens previamente curadas pelo *FGAP*, sendo que os melhores resultados foram obtidos nas montagens geradas pelos montadores SGA e *AbySS* para o organismo *S. aureus*, onde o *GapBlaster* conseguiu fechar até 5 *gaps* que não foram previamente identificados pelo *FGAP*.

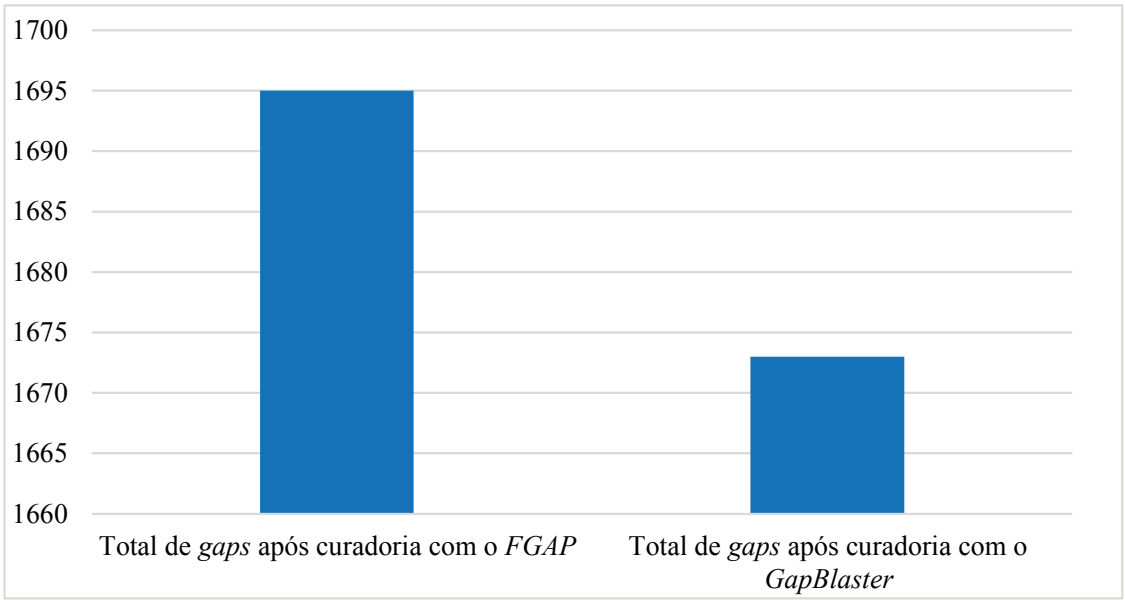
Gráfico 3 – Melhoria obtida nas montagens previamente curadas pelo *FGAP*



Fonte: Dados obtidos de Sá et al. (2016)

Contudo, quando consideradas todas as montagens curadas subsequentemente pelo *GapBlaster*, são totalizados 22 *gaps* fechados que não foram previamente identificados pelo *FGAP*, como demonstrado no Gráfico 4. Para maiores detalhes sobre os resultados da análise de uso do *GapBlaster* em conjunto com o *FGAP*, ver Tabela 5 do Apêndice A.

Gráfico 4 – Total de *gaps* restantes após utilizar o *GapBlaster* nas montagens previamente curadas pelo *FGAP*

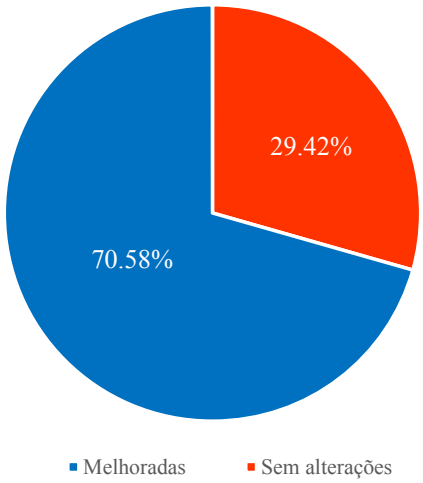


Fonte: Dados obtidos de Sá et al. (2016)

Com base nos resultados desta análise, presume-se que além do *GapBlaster* facilitar o processo de fechamento de *gaps* através da sua interface gráfica, ele também pode ser utilizado em conjunto com outras ferramentas para acelerar ainda mais o processo de curadoria.

Para confirmar os resultados alcançados, foi avaliado se o *GapBlaster* também poderia melhorar a saída gerada pelo *GapFiller* nos dados obtidos do projeto GAGE. No Gráfico 5, é demonstrado que o *GapBlaster* diminuiu a quantidade de *gaps* em 70.58% das montagens do projeto GAGE previamente curadas pelo *GapFiller*.

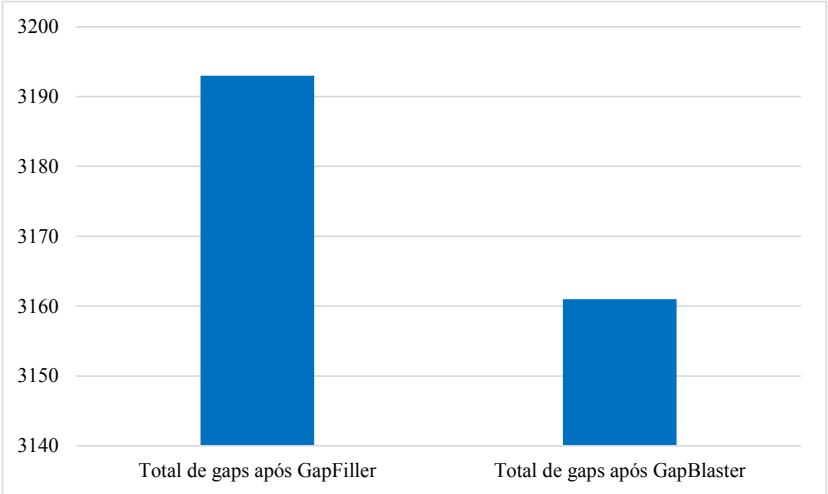
Gráfico 5 – Melhoria obtida nas montagens do projeto GAGE previamente curadas pelo *GapFiller*



Fonte: Dados obtidos de Sá et al. (2016)

Novamente, os melhores resultados foram obtidos no fechamento de *gaps* dos dados gerados pelos montadores *ABySS2* e SGA para o organismo *S. aureus*, onde os *gaps* foram reduzidos de 35 para 30 e de 654 para 646, respectivamente (ver Tabela 6 do Apêndice A). Contudo, a contagem geral revela que ao todo 32 *gaps* foram fechados nas montagens que foram aprimoradas pelo *GapBlaster*, conforme exibido no Gráfico 6.

Gráfico 6 – Total de *gaps* restantes após utilizar o *GapBlaster* nas montagens previamente curadas pelo *GapFiller*



Fonte: Dados obtidos de Sá et al. (2016)

Desta forma, é possível verificar que de fato o *GapBlaster* é um valioso programa de código livre que pode ser utilizado em conjunto com outras ferramentas de fechamento de *gap* para produzir montagens de genomas mais completas.

4.4 Precisão dos Testes Realizados

A exatidão das bases inseridas nos *gaps* que foram fechados foi avaliada através do alinhamento dos *scaffolds* originais e todos os resultados produzidos pelo *GapBlaster*, *FGAP* e *GapFiller* contra os genomas de referência descritos na Tabela 2. Os resultados disponíveis nas Tabelas 7–9, disponíveis no Apêndice B, mostram que todos os arquivos produzidos na análise de fechamento de *gaps* obtiveram porcentagens de alinhamento similares aos arquivos originais, o que confirma que as bases inseridas nos *gaps* estavam corretas.

Apesar do *GapBlaster* ter sido implementado para funcionar com os alinhadores *Legacy Blast*, *Blast+* e *MUMmer*, somente foi utilizado o *Blast+* na análise, pois este é o mesmo alinhador utilizado pelo programa *FGAP*. Entretanto, todos os três alinhadores foram testados nos dados do projeto GAGE, e foi possível observar que o *Legacy Blast* e *Blast+* apresentaram resultados similares (ver Tabelas 10–11 no Apêndice C).

4.5 Comparação das Funcionalidades

Por último, foram feitas comparações entre as funcionalidades dos programas *GapBlaster*, *FGAP* e *GapFiller* no Quadro 1. Estas comparações ajudaram a identificar a principal vantagem do *GapBlaster*, que é a interface gráfica que permite utilizar *contigs* para realizar curadoria manual e fechar *gaps* com maior precisão.

Quadro 1 – Comparação das funcionalidades do *GapBlaster*, *FGAP* e *GapFiller*

Funcionalidades	***GapBlaster***	***FGAP***	***GapFiller***
Método de alinhamento	*Legacy Blast*, *Blast+* ou *MUMmer*	*Blast+*	*Bowtie* ou BWA
Ajuste do tamanho da região flanqueadora	Sim	Sim	Sim
Permite curadoria manual	Sim	Não	Não
Realiza análise automática	Sim	Sim	Sim
Usa leituras pareadas para fechar *gaps*	Não	Não	Sim
Usa *contigs* para fechar *gaps*	Sim	Sim	Não
Lê arquivos nos formatos FASTQ, SAM e BAM	Não	Não	Sim
Executa código em paralelo	Não	Sim	Não
Interface gráfica	Sim	Não	Não
Melhora o resultado de outros programas	Sim	Não foi testado	Não foi testado
Fecha *gaps* corretamente	Sim	Sim	Sim

Fonte: Sá et al. (2016)

5 CONSIDERAÇÕES FINAIS

Apesar do *GapFiller* utilizar leituras pareadas, é possível verificar na Tabela 4 que o *GapBlaster* apresentou aumento significativo na quantidade de *gaps* fechados ao mapear *contigs* contra *scaffolds*. Alternativamente, o *GapBlaster* pode ser utilizado em conjunto com outros programas de fechamento de *gap* para agilizar a completude do genoma através da manipulação manual, como foi demonstrado na análise em que o *GapBlaster* foi usado para melhorar os resultados já obtidos pelos programas *FGAP* e *GapFiller*.

Assim sendo, ainda que se ganhe tempo ao utilizar ferramentas que fechem *gaps* automaticamente, a curadoria manual auxilia na aquisição de resultados mais precisos, tais como os apresentados pelo *GapBlaster* nas Tabelas 7–9 do Apêndice B. Adicionalmente, foi constatado que a utilização do *software* por terceiros é simplificada através do uso de interface gráfica.

Desta forma, é possível concluir que o programa *GapBlaster* tem a vantagem de introduzir menos erros na curadoria do genoma, baseado na habilidade da interface gráfica de permitir que o usuário decida se um *gap* está sendo preenchido corretamente.

5.1 Publicação do Trabalho

Os resultados deste trabalho foram publicados no formato de artigo científico no periódico acadêmico *PLOS ONE*, que é de acesso aberto e possui veiculação internacional. Em 2015, o fator de impacto deste periódico foi de 3.057 e o mesmo teve classificação B2 no Qualis da CAPES para a área de Ciência da Computação. A versão final do artigo, que está escrita em inglês, foi disponibilizada no Anexo A desta monografia.

5.2 Manual do Usuário

Adicionalmente, foi disponibilizado no Anexo B o manual do usuário do *GapBlaster*, que também está em inglês. Este manual contém informações detalhadas sobre os requisitos de sistema, bem como instruções de instalação e execução do *software*.

5.3 Trabalhos Futuros

Uma das melhorias que podem ser feitas no trabalho diz respeito ao manual do usuário. Por exemplo, a versão atual do manual somente contém instruções de instalação dos pré-requisitos no sistema operacional *Debian* e seria interessante que também houvessem instruções para outros sistemas operacionais, como *Windows* e *Mac OS*. Outra melhoria que poderia ser feita no manual do usuário seria o uso do *LaTeX* na elaboração do mesmo e a utilização de imagens em formato vetorial, para assim evitar a "pixelização"em monitores com alta resolução.

Em relação à divulgação do trabalho, uma possível melhoria seria a utilização do *GitHub* para hospedar a página de *download* do programa e também o código fonte. Desta forma, o programa ganharia mais utilizadores e consequentemente mais citações.

Por último, o desempenho do *GapBlaster* pode ser melhorado através da paralelização do código. Desta forma, o programa poderá utilizar mais de 1 núcleo de processamento e assim agilizar algumas etapas do algoritmo 1, como a conversão, a ordenação e a concatenação de alinhamentos. Esta concorrência pode ser implementada através da utilização da classe *Thread* disponibilizada no *Java*, por exemplo.

Acredita-se que com essas melhorias o *GapBlaster* possa gerar resultados mais rápidos e que a ferramenta se torne mais acessível aos usuários.

REFERÊNCIAS

AIRD, D. et al. Analyzing and minimizing pcr amplification bias in illumina sequencing libraries. **Genome biology**, BioMed Central, v. 12, n. 2, p. R18, 2011.

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

BOETZER, M.; PIROVANO, W. Toward almost closed genomes with gapfiller. **Genome biology**, BioMed Central, v. 13, n. 6, p. R56, 2012.

DORELLA, F. A. et al. Corynebacterium pseudotuberculosis: microbiology, biochemical properties, pathogenesis and molecular studies of virulence. **Veterinary research**, EDP Sciences, v. 37, n. 2, p. 201–218, 2006.

FRASER, C. M. et al. The value of complete microbial genome sequencing (you get what you pay for). **Journal of bacteriology**, Am Soc Microbiol, v. 184, n. 23, p. 6403–6405, 2002.

MARDIS, E. R. The impact of next-generation sequencing technology on genetics. **Trends in genetics**, Elsevier, v. 24, n. 3, p. 133–141, 2008.

MARDIS, E. R. Next-generation sequencing platforms. **Annual review of analytical chemistry**, Annual Reviews, v. 6, p. 287–303, 2013.

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

MOUNT, D. W. **Bioinformatics: Sequence and Genome Analysis**. Cold Spring Harbor Laboratory Press, 2004. (G - Reference,Information and Interdisciplinary Subjects Series). ISBN 9780879697129. Disponível em: <https://books.google.com.br/books?id=bvY21DGa1OwC>.

MUKHERJEE, S. et al. Genomes online database (gold) v. 6: data updates and feature enhancements. **Nucleic Acids Research**, Oxford Univ Press, p. gkw992, 2016.

NAGARAJAN, N. et al. Finishing genomes with limited resources: lessons from an ensemble of microbial genomes. **BMC genomics**, BioMed Central, v. 11, n. 1, p. 1, 2010.

NAGARAJAN, N.; POP, M. Sequencing and genome assembly using next-generation technologies. **Computational Biology**, Springer, p. 1–17, 2010.

PIRO, V. C. et al. Fgap: an automated gap closing tool. **BMC research notes**, BioMed Central, v. 7, n. 1, p. 371, 2014.

REIS-FILHO, J. S. Next-generation sequencing. **Breast Cancer Research**, BioMed Central, v. 11, n. 3, p. S12, 2009.

SÁ, P. H. de et al. Gapblaster: a graphical gap filler for prokaryote genomes. **PloS one**, Public Library of Science, v. 11, n. 5, p. e0155327, 2016.

SALZBERG, S. L. et al. Gage: a critical evaluation of genome assemblies and assembly algorithms. **Genome research**, Cold Spring Harbor Lab, v. 22, n. 3, p. 557–567, 2012.

SANGER, F.; BROWNLEE, G.; BARRELL, B. A two-dimensional fractionation procedure for radioactive nucleotides. **Journal of molecular biology**, Elsevier, v. 13, n. 2, p. 373IN1–398IN4, 1965.

SANGER, F.; NICKLEN, S.; COULSON, A. R. Dna sequencing with chain-terminating inhibitors. **Proceedings of the national academy of sciences**, National Acad Sciences, v. 74, n. 12, p. 5463–5467, 1977.

SCHUSTER, S. C. Next–generation sequencing transforms today's biology. **Nature**, v. 200, n. 8, p. 16–18, 2007.

SELKOV, E. et al. Functional analysis of gapped microbial genomes: amino acid metabolism of thiobacillus ferrooxidans. **Proceedings of the National Academy of Sciences**, National Acad Sciences, v. 97, n. 7, p. 3509–3514, 2000.

SOUEIDAN, H. et al. Finishing bacterial genome assemblies with mix. **BMC bioinformatics**, BioMed Central Ltd, v. 14, n. Suppl 15, p. S16, 2013.

WATSON, J. D.; CRICK, F. H. The structure of dna. In: COLD SPRING HARBOR LABORATORY PRESS. **Cold Spring Harbor symposia on quantitative biology**. [S.l.], 1953. v. 18, p. 123–131.

WOJCIESZEK, M. et al. Genomes correction and assembling: present methods and tools. In: INTERNATIONAL SOCIETY FOR OPTICS AND PHOTONICS. **Symposium on Photonics Applications in Astronomy, Communications, Industry and High-Energy Physics Experiments**. [S.l.], 2014. p. 92901X–92901X.

Apêndices

APÊNDICE A – RESULTADOS OBTIDOS AO USAR O *GAPBLASTER* EM CONJUNTO COM OUTROS PROGRAMAS DE CURADORIA

Tabela 5 – Resultados obtidos ao usar o *GapBlaster* em conjunto com o *FGAP*

Organismo	Montador	#Gaps *FGAP*	#N *FGAP*	#Gaps após GB	#N após GB
S. aureus A–S391_USA300					
	AbySS	45	51127	41	45439
	MSR-CA	47	7861	46	6359
	SGA	634	298252	629	290825
R. sphaeroides 2.4.1					
	AbySS2	228	60323	227	60040
	Allpaths-LG	82	19500	81	19494
	Bambus2	80	55990	79	55402
	CABOG	190	21065	188	19568
	MSR-CA	347	31174	343	25592
	SOAPdenovo	37	10097	36	9237
C. pseudotuberculosis 262					
	SPADES	5	360	3	251

Fonte: Sá et al. (2016)

Tabela 6 – Resultados obtidos do *GapBlaster* em conjunto com o *GapFiller*

Organismo	Montador	#Gaps GF	#N GF	#Gaps após GB	#N após GB
S. aureus A–S391_USA300					
	AbySS	69	56355	66	54837
	AbySS2	35	10003	30	8741
	Allpaths-LG	40	10472	39	10455
	Bambus2	98	30771	97	30725
	MSR-CA	80	11651	76	9794
	SGA	654	312284	646	307095
R. sphaeroides 2.4.1					
	AbySS	306	118298	304	118287
	AbySS2	290	68052	288	67740
	Allpaths-LG	164	24001	163	23780
	CABOG	191	25011	190	24336
	MSR-CA	336	37494	333	33590
	SGA	930	1159235	929	1159162

Fonte: Sá et al. (2016)

APÊNDICE B – PERCENTUAL DE BASES ALINHADAS

Tabela 7 – Percentual de bases alinhadas para o organismo *S. aureus*

Staphylococcus aureus	**Total de Bases**	**Bases Alinhadas**	**Bases não Alinhadas**	**Média de Comprimento**	**Semelhança Média**
Abyss – Original	3893185	3637985(93.44%)	255200(6.56%)	6554.28	99.97
Abyss – GB	3891853	3644836(93.65%)	247017(6.35%)	6667.18	99.96
Abyss – *FGAP*	3892419	3641908(93.56%)	250511(6.44%)	6793.26	99.96
Abyss – GF	3892385	3636721(93.43%)	255664(6.57%)	6538.14	99.97
Abyss2 – Original	3821622	3763970(98.49%)	57652(1.51%)	24849.97	99.98
Abyss2 – GB	3821547	3765506(98.53%)	56041(1.47%)	25958.54	99.98
Abyss2 – *FGAP*	3820728	3767543(98.61%)	53185(1.39%)	27926.76	99.98
Abyss2 – GF	3821622	3763296(98.47%)	58326(1.53%)	24632.06	99.98
Allpaths-LG – Original	2880676	2848037(98.87%)	32639(1.13%)	39022.08	99.98
Allpaths-LG – GB	2880927	2848697(98.88%)	32230(1.12%)	40704.4	99.98
Allpaths-LG – *FGAP*	2880402	2848864(98.91%)	31538(1.09%)	43175.98	99.97
Allpaths-LG – GF	2880126	2846976(98.85%)	33150(1.15%)	41878.63	99.98
Bambus2 – Original	2862930	2808616(98.10%)	54314(1.90%)	9824.04	99.99
Bambus2 – GB	2862962	2808690(98.10%)	54272(1.90%)	9824.94	99.99
Bambus2 – *FGAP*	2862227	2809501(98.16%)	52726(1.84%)	9999.39	99.98
Bambus2 – GF	2862871	2807087(98.05%)	55784(1.95%)	9852.53	99.99
MSR-CA – Original	2872905	2862223(99.63%)	10682(0.37%)	23562.38	99.97
MSR-CA – GB	2872603	2864405(99.71%)	8198(0.29%)	24787.72	99.97
MSR-CA – *FGAP*	2873324	2864955(99.71%)	8369(0.29%)	29365.24	99.96
MSR-CA – GF	2872831	2860957(99.59%)	11874(0.41%)	23532.99	99.97
SGA – Original	3128388	2805042(89.66%)	323346(10.34%)	2526.83	99.91
SGA – GB	3123055	2808249(89.92%)	314806(10.08%)	2555.02	99.9
SGA – *FGAP*	3126702	2805382(89.72%)	321320(10.28%)	2543.38	99.9
SGA – GF	3128388	2794559(89.33%)	333829(10.67%)	2512.8	99.86
SOAPdenovo – Original	2924135	2891010(98.87%)	33125(1.13%)	18281.16	99.96
SOAPdenovo – GB	2924134	2891029(98.87%)	33105(1.13%)	18281.33	99.96
SOAPdenovo – *FGAP*	2924221	2891245(98.87%)	32976(1.13%)	18281.72	99.96
SOAPdenovo – GF	2924135	2890886(98.86%)	33249(1.14%)	18280.5	99.96
Velvet – Original	2877995	2829844(98.33%)	48151(1.67%)	12989.11	99.97
Velvet – GB	2878155	2830219(98.33%)	47936(1.67%)	13232.53	99.97
Velvet – *FGAP*	2879170	2830521(98.31%)	48649(1.69%)	15293.98	99.97
Velvet – GF	2877904	2827668(98.25%)	50236(1.75%)	12974.1	99.97

Fonte: Sá et al. (2016)

Tabela 8 – Percentual de bases alinhadas para o organismo *R. sphaeroides*

Rhodobacter sphaeroides	**Total de Bases**	**Bases Alinhadas**	**Bases não Alinhadas**	**Média de Comprimento**	**Semelhança Média**
Abyss – Original	5160167	5012070(97.13%)	148097(2.87%)	2003.33	99.93
Abyss – GB	5160167	5012070(97.13%)	148097(2.87%)	2003.33	99.93
Abyss – *FGAP*	5160222	5012145(97.13%)	148077(2.87%)	2006.37	99.93
Abyss – GF	5157773	4998867(96.92%)	158906(3.08%)	2005.52	99.89
Abyss2 – Original	5331930	5269443(98.83%)	62487(1.17%)	7272.37	99.96
Abyss2 – GB	5331648	5269603(98.84%)	62045(1.16%)	7288.38	99.96
Abyss2 – *FGAP*	5332026	5271786(98.87%)	60240(1.13%)	7332.84	99.96
Abyss2 – GF	5331786	5264536(98.74%)	67250(1.26%)	7253.12	99.95
Allpaths-LG – Original	4609785	4588162(99.53%)	21623(0.47%)	21602.84	99.99
Allpaths-LG – GB	4609792	4588765(99.54%)	21027(0.46%)	21812	99.99
Allpaths-LG – *FGAP*	4609490	4589696(99.57%)	19794(0.43%)	22568.98	99.99
Allpaths-LG – GF	4609504	4585835(99.49%)	23669(0.51%)	21695.67	99.98
Bambus2 – Original	4428612	4371494(98.71%)	57118(1.29%)	8052.3	99.99
Bambus2 – GB	4428840	4372310(98.72%)	56530(1.28%)	8068.67	99.99
Bambus2 – *FGAP*	4428433	4372126(98.73%)	56307(1.27%)	8083.22	99.99
Bambus2 – GF	4428710	4370562(98.69%)	58148(1.31%)	8065.46	100
CABOG – Original	4259679	4238284(99.50%)	21395(0.50%)	13053.88	99.96
CABOG – GB	4259686	4238946(99.51%)	20740(0.49%)	13096.22	99.96
CABOG – *FGAP*	4259669	4238652(99.51%)	21017(0.49%)	13135.85	99.96
CABOG – GF	4259565	4235201(99.43%)	24364(0.57%)	13004.91	99.95
MSR-CA – Original	4498559	4464796(99.25%)	33763(0.75%)	11302.87	99.97
MSR-CA – GB	4512606	4485278(99.39%)	27328(0.61%)	11430.54	99.96
MSR-CA – *FGAP*	4498115	4465439(99.27%)	32676(0.73%)	11480.69	99.96
MSR-CA – GF	4497999	4459944(99.15%)	38055(0.85%)	11780.9	99.96
SGA – Original	5614693	4468485(79.59%)	1146208(20.41%)	1489.96	99.98
SGA – GB	5614693	4468485(79.59%)	1146208(20.41%)	1489.96	99.98
SGA – *FGAP*	5614783	4468574(79.59%)	1146209(20.41%)	1491.3	99.97
SGA – GF	5614760	4455655(79.36%)	1159105(20.64%)	1487.62	99.97
SOAPdenovo – Original	4627058	4616325(99.77%)	10733(0.23%)	7499.26	99.97
SOAPdenovo – GB	4626764	4616879(99.79%)	9885(0.21%)	7500.17	99.97
SOAPdenovo – *FGAP*	4626784	4616415(99.78%)	10369(0.22%)	7560.41	99.97
SOAPdenovo – GF	4627058	4615650(99.75%)	11408(0.25%)	7486.09	99.97
Velvet – Original	4615068	4527551(98.10%)	87517(1.90%)	6239.01	99.97
Velvet – GB	4615335	4527838(98.10%)	87497(1.90%)	6248.35	99.97
Velvet – *FGAP*	4615556	4528462(98.11%)	87094(1.89%)	6395.91	99.96
Velvet – GF	4614833	4520446(97.95%)	94387(2.05%)	6348.65	99.95

Fonte: Sá et al. (2016)

Tabela 9 – Percentual de bases alinhadas para o organismo *C. pseudotuberculosis*

Corynebacterium pseudotuberculosis	**Total de Bases**	**Bases Alinhadas**	**Bases não Alinhadas**	**Média de Comprimento**	**Semelhança Média**
Original	2893857	2412989(83.38%)	480868(16.62%)	6419.58	99.97
GapBlaster	2893146	2413130(83.41%)	480016(16.59%)	6834.41	99.97
FGAP	2892817	2413359(83.43%)	479458(16.57%)	7084.6	99.97

Fonte: Sá et al. (2016)

APÊNDICE C – COMPARAÇÃO DOS ALINHADORES

Tabela 10 – Comparação dos alinhadores usados no GapBlaster para o organismo *S. aureus*

Staphylococcus aureus	**Total de Alinhamentos**	**Gaps Fechados**	**Ns Reduzidos**
Abyss – MUMmer	Erro	Erro	Erro
Abyss – Legacy Blast	14	5	3454
Abyss – Blast+	10	4	1653
Abyss2 – MUMmer	Erro	Erro	Erro
Abyss2 – Legacy Blast	11	9	2303
Abyss2 – Blast+	13	4	925
AllPathsLG – MUMmer	74	1	1
AllPathsLG – Legacy Blast	110	1	51
AllPathsLG – Blast+	107	1	1
Bambus2 – MUMmer	8	1	32
Bambus2 – Legacy Blast	21	2	2448
Bambus2 – Blast+	11	1	33
MSR-CA – *MUMmer*	33	3	89
MSR-CA – *Legacy Blast*	48	5	2227
MSR-CA – *Blast+*	33	5	2060
SGA – *MUMmer*	33	3	89
SGA – *Legacy Blast*	48	5	2227
SGA – *Blast+*	33	5	2060
SOAPdenovo – MUMmer	Erro	Erro	Erro
SOAPdenovo – Legacy Blast	14	1	20
SOAPdenovo – Blast+	17	1	20
Velvet – MUMmer	Erro	Erro	Erro
Velvet – Legacy Blast	15	2	195
Velvet – Blast+	14	2	195

Fonte: Sá et al. (2016)

Tabela 11 – Comparação dos alinhadores usados no GapBlaster para o organismo *R. sphaeroides*

Rhodobacter sphaeroides	**Total de Alinhamentos**	**Gaps Fechados**	**Ns Reduzidos**
Abyss – MUMmer	Erro	Erro	Erro
Abyss – Legacy Blast	5	2	408
Abyss – Blast+	4	0	0
Abyss2 – MUMmer	Erro	Erro	Erro
Abyss2 – Legacy Blast	7	3	996
Abyss2 – Blast+	5	2	1126
AllPathsLG – MUMmer	11	3	1194
AllPathsLG – Legacy Blast	9	2	521
AllPathsLG – Blast+	15	2	434
Bambus2 – MUMmer	5	1	51
Bambus2 – Legacy Blast	6	1	51
Bambus2 – Blast+	4	1	51
CABOG – MUMmer	4	2	984
CABOG – Legacy Blast	3	2	984
CABOG – Blast+	4	2	984
MSR-CA – *MUMmer*	24	4	4132
MSR-CA – *Legacy Blast*	20	3	3248
MSR-CA – *Blast+*	19	5	4394
SGA – *MUMmer*	Erro	Erro	Erro
SGA – *Legacy Blast*	0	0	0
SGA – *Blast+*	0	0	0
SOAPdenovo – MUMmer	Erro	Erro	Erro
SOAPdenovo – Legacy Blast	2	1	860
SOAPdenovo – Blast+	3	1	860
Velvet – MUMmer	Erro	Erro	Erro
Velvet – Legacy Blast	17	2	20
Velvet – Blast+	19	2	20

Fonte: Sá et al. (2016)

Anexos

ANEXO A – ARTIGO PUBLICADO

PLOS ONE

RESEARCH ARTICLE

GapBlaster—A Graphical Gap Filler for Prokaryote Genomes

Pablo H. C. G. de Sá[1☯], Fábio Miranda[1☯], Adonney Veras[1], Diego Magalhães de Melo[1], Siomar Soares[2], Kenny Pinheiro[1], Luis Guimarães[1], Vasco Azevedo[3], Artur Silva[1], Rommel T. J. Ramos[1]*

1 Institute of Biological Sciences, Federal University of Pará, Belém, Pará, Brazil, **2** Institute of Biological and Natural Sciences, Federal University of Triângulo Mineiro, Uberaba, Minas Gerais, Brazil, **3** Institute of Biological Sciences, Federal University of Minas Gerais, Belo Horizonte, Minas Gerais, Brazil

☯ These authors contributed equally to this work.
* rommelthiago@gmail.com

Citation: de Sá PHCG, Miranda F, Veras A, de Melo DM, Soares S, Pinheiro K, et al. (2016) GapBlaster—A Graphical Gap Filler for Prokaryote Genomes. PLoS ONE 11(5): e0155327. doi:10.1371/journal.pone.0155327

Editor: I. King Jordan, Georgia Institute of Technology, UNITED STATES

Received: January 6, 2016

Accepted: April 27, 2016

Published: May 12, 2016

Data Availability Statement: All relevant data are within the paper and its Supporting Information files.

Funding: RTJR, AS, and VA are supported by Biologia Computacional Coordenação de Aperfeiçoamento de Pessoal de Nível Superior N° 051/ 2013 and Conselho Nacional de Desenvolvimento Científico e Tecnológico grant #482799/2013-7.

Competing Interests: The authors have declared that no competing interests exist.

Abstract

The advent of NGS (Next Generation Sequencing) technologies has resulted in an exponential increase in the number of complete genomes available in biological databases. This advance has allowed the development of several computational tools enabling analyses of large amounts of data in each of the various steps, from processing and quality filtering to gap filling and manual curation. The tools developed for gap closure are very useful as they result in more complete genomes, which will influence downstream analyses of genomic plasticity and comparative genomics. However, the gap filling step remains a challenge for genome assembly, often requiring manual intervention. Here, we present GapBlaster, a graphical application to evaluate and close gaps. GapBlaster was developed via Java programming language. The software uses contigs obtained in the assembly of the genome to perform an alignment against a draft of the genome/scaffold, using BLAST or Mummer to close gaps. Then, all identified alignments of contigs that extend through the gaps in the draft sequence are presented to the user for further evaluation via the GapBlaster graphical interface. GapBlaster presents significant results compared to other similar software and has the advantage of offering a graphical interface for manual curation of the gaps. GapBlaster program, the user guide and the test datasets are freely available at https://sourceforge.net/projects/gapblaster2015/. It requires Sun JDK 8 and Blast or Mummer.

Introduction

Next generation sequencing (NGS) platforms have reduced sequencing costs and increased the amount of data generated, resulting in a greater number of complete genomes for eukaryotes and prokaryotes, which are subsequently deposited in public databases [1,2].

Several computational tools have been developed for processing reads, such as error correction and quality filters, as well as additional programs and pipelines that perform genome assemblies of reads generated by NGS platforms, producing complete genomes or scaffolds

[3,4]. As a result of assembly reads, many contigs are produced. These reads or reference genomes can be used to order the contigs to produce a scaffold. Some regions in the scaffold have no assigned bases (A,C,T or G) due to the limitations of sequencing technology or assembly algorithms; these regions are called gaps and are usually represented by Ns [5–7].

Beyond commercial programs, such as CLC Genomic Workbench and Lasergene Suite, which have available options for finishing genome assemblies, including steps that fill gaps, open source programs are available. For example the open source programs G4ALL [8], GapCloser [3], GapFiller [6], and FGAP [9] use different approaches, such as paired reads or results of assemblies obtained with different software, to fill gap regions. The FGAP program was implemented in Matlab language and uses a draft of the assembly and a set of contigs that are mapped against genome draft to close gaps using BLAST algorithms. Both a fasta and a log file that report the filled gaps are generated at the end of the process. However, FGAP has no graphical interface [9].

G4ALL was implemented via JAVA programming language. The software has a graphical interface that allows the user to perform gap closure through manual curation of the scaffolds by comparing the BLAST results of the assembled contigs to the assembled scaffolds, similar to the GapBlaster method. G4ALL is useful for extending the contigs based on the overlap between them; however, it does not use contigs to close the gap regions [8].

GapCloser uses the information from paired reads to extend the sequences of contigs between gaps. Thus, the gaps can be closed or reduced [3]. Similar to GapCloser, the GapFiller program uses paired reads and is able to use data from different sequencing rounds simultaneously [6]. It is one of the available tools for closing gaps in prokaryotic and eukaryotic genomes of sizes up to ~100 Mb [10].

Genomes that have gaps may impair further studies because they may only partially represent an organism's gene repertoire. Incomplete genomes can affect downstream analyses of genomic plasticity and comparative genomics [11].

Therefore, it is important to use complete genomes for comparative studies to properly characterize genome structure variations and gene content. This characterization allows the identification of genes that are 1) shared among all isolates and are thus useful for applied issues, such as vaccine and drug design [12]; 2) shared by some organisms, but not all studied organisms, and are thus useful for studying the reference lab activities for pathogenic bacteria [13,14]; and 3) present in a single isolate providing information regarding bacteria lifestyle [15].

Thus, this study presents a computational tool with a graphical user interface that helps reduce gaps through manual curation to increase the completion of genome assembly, rather than relying on the complete automation of this task.

Materials and Methods

Implementation

The GapBlaster was developed via JAVA programming language (http://java.sun.com/) using the paradigm of object orientation and the Swing library to create the visual resources (http://java.sun.com/docs/books/tutorial/uiswing).

Through the main interface of GapBlaster (S1 Fig), the user can input the scaffold and the contig files in FASTA format. After processing, another screen (S2 Fig) shows the alignment results. The user is then able to perform manual curation and select alignments that fill gaps confidently, as when the user finds a contig aligned in the gap flanks, closing the gap completely, as shown in S3 Fig.

GapBlaster performs five steps to identify possible gaps to be filled. All of the contigs obtained in the assembly are aligned against the draft genome or scaffold using BLAST Legacy

[16], Blast+ or Mummer [17] based on user choice, and the alignment result is converted to the GapBlaster format. The contigs are subsequently ordered according to the mapping position in the scaffold. The program searches the alignments of the same contig that flank gap regions. A new ordination of the alignments is performed to determine the best option for gap closure. All identified alignments that fill gaps are presented to the user for evaluation (accepted or rejected) through the GapBlaster interface, and a log of changes made is generated. The selection of the alignment and the parameters can be defined by the user through the GapBlaster interface.

Test data

To evaluate the GapBlaster program, analyses were conducted using two datasets: the first used sequencing data of *Corynebacterium pseudotuberculosis*, and the second was obtained from the GAGE (Genome Assembly Gold-Standard Evaluation) assembly of genomes [18].

C. pseudotuberculosis is a facultative intracellular gram-positive bacterium that causes caseous lymphadenitis (CLA), an infectious disease that affects small ruminants and belongs to CMNR group (Corynebacterium, Mycobacterium, Nocardia, and Rhodococcus) [19].

The sequencing of *C. pseudotuberculosis* was performed by an Ion Torrent PGM platform (Table 1). The reads (available in SRA database: SRR3312980) were assembled by a *de novo* strategy using SPADES version 3.1.0, with default parameters for Ion Torrent PGM data [20]. The scaffolds and contigs files produced in the assembly (available in https://sourceforge.net/projects/gapblaster2015/files/test_dataset/) were used as inputs in GapBlaster.

The GAGE dataset had the assemblies of the *Staphylococcus aureus* and *Rhodobacter sphaeroides* genomes, containing contigs and scaffolds generated by the following assemblers: Abyss, ABySS2, AllPaths-LG, Bambus2, MSR-CA, SGA, SOAPdenovo and Velvet for both organisms, whereas the CABOG was used for only *Rhodobacter sphaeroides* [18]. The data are available at http://gage.cbcb.umd.edu/, and the genome sequencing information can be seen in Table 1.

GapBlaster

All contigs and scaffolds of the datasets were manually evaluated with GapBlaster version 1.1.1 to close gaps. In our analysis, we used one scaffold and one contig file for each organism/assembly, with the parameter Flank Length = 11 and the aligner Blast+ (the parameters in the GapBlaster should be set to reproduce our results). To close gaps, regions flanking the gaps (represented by Ns) were considered only when they had high identity (the threshold should be defined by the user).

Gap closure comparison

To compare gap filling performance, GapBlaster, GapFiller and FGAP software were used in a gap closure analysis of the GAGE dataset and *C. pseudotuberculosis*.

Table 1. Sequencing information of the genomes used in the analysis.

Organism	Platform	Library	Read Length	Insert Size	Number of Reads
Corynebacterium pseudotuberculosis 262	Ion Torrent PGM	Fragment	~220 bp	N/A	1765213
Staphylococcus aureus A-S391_USA300	Illumina	Paired-end	~101 bp	180 bp	1294104
Staphylococcus aureus A-S391_USA300	Illumina	Mate-Pair	~37 bp	3500 bp	3494070
Rhodobacter sphaeroides 2.4.1	Illumina	Paired-end	~101 bp	180 bp	2050868
Rhodobacter sphaeroides 2.4.1	Illumina	Mate-Pair	~101 bp	3500 bp	2050868

doi:10.1371/journal.pone.0155327.t001

The GAGE dataset with the mate-pair reads was analyzed with GapFiller [6] and FGAP [9] based on gap closure performance. Both types of software were used under default parameters, and the results were subsequently compared to GapBlaster.

The *C. pseudotuberculosis* genome was analyzed with FGAP only as GapFiller software requires paired-end libraries, and *C. pseudotuberculosis* was sequenced using fragmented libraries. The results of FGAP were compared to GapBlaster.

Additionally, GapBlaster was used to reduce gaps in the output files of FGAP and GapFiller software.

Results evaluation

To validate the gap filling analysis, an in-house script was developed to evaluate the amount of gaps and Ns for each of the tests. The FASTA file (original scaffolds and the results of GapBlaster, FGAP, and GapFiller) was used as an input to count the number of gaps and their respective sizes. This script and a brief manual are available at https://sourceforge.net/projects/gapblaster2015/upload/scripts/.

To confirm if the gaps were correctly closed, the validation script of GAGE was used (http://gage.cbcb.umd.edu/results/gage-paper-validation.tar.gz). The input of this script was the reference genome (Table 2) and the original scaffold or gap-filled scaffold file.

Results and Discussion

The assembly results (number of bases and scaffolds) of the *C. pseudotuberculosis* genome produced by SPADES and the information concerning several assemblies of *S. aureus* and *R. sphaeroides* produced by various types of assemblers are shown in Table 3.

The results of the gap closure process for the *Corynebacterium* data assembled by SPADES are shown in Table 4.

For *C. pseudotuberculosis* the amount of gaps was reduced from 24 to 11 with GapBlaster, and from 24 to 5, with FGAP. Gap length was also reduced for the *Corynebacterium* genome, as shown in Table 4. The *C. pseudotuberculosis* genome was sequenced using fragment libraries; thus, they could not be submitted to GapFiller.

The GAGE data of *S. aureus* and *R. sphaeroides* were assembled by several assemblers, and the results (contigs and scaffolds) were submitted to GapBlaster, FGAP and GapFiller. All assemblies of *S. aureus* revealed reductions in gaps and Ns when analyzed by GapBlaster. For *R. sphaeroides*, only the data for SGA did not show a reduction in gaps by GapBlaster (Table 5). It is important to highlight that GapBlaster allows manual curation; it allows less stringent criteria with careful manual evaluation, which is able to produce better results.

The FGAP and GapFiller programs were used to perform the gap closure step, and these results were compared with those obtained by GapBlaster (Table 5). GapFiller increased the

Table 2. Information of the reference genomes used to validate the filled-in gaps.

Organism	*Corynebacterium Pseudotuberculosis 262*	*Staphylococcus Aureus* A-S391_USA300	*Rhodobacter Sphaeroides 2.4.1*
Genome Size	2325749	2872916	4628173
GC Content	52,17	0,3276	68,77
Number of Chrs	1	1	2
Number of Plasmids	0	0	5
Genbank	CP012022.1	CP007690.1	GCA_000273405.1
150 pb Repeats	8007	10709	38073
250 pb Repeats	6612	9460	35353

doi:10.1371/journal.pone.0155327.t002

Table 3. Genome assembly information for *C. pseudotuberculosis* 262, *S. aureus* and *R. sphaeroides*.

Organism	Assembler	Bases (with N)	#Scaffolds
C. pseudotuberculosis 262	- - - - - - -	- - - - - - -	- - - - - - -
	SPADES	2893857	4611
S. aureus	- - - - - - -	- - - - - - -	- - - - - - -
	ABySS	3893185	5012
	ABySS2	3821622	125
	Allpaths-LG	2880676	19
	Bambus2	2862930	17
	MSR-CA	2872905	17
	SGA	3128388	546
	SOAPdenovo	2924135	175
	Velvet	2877995	173
R. sphaeroides	- - - - - - -	- - - - - - -	- - - - - - -
	ABySS	5160167	2714
	ABySS2	5331930	480
	Allpaths-LG	4609785	38
	Bambus2	4428612	92
	CABOG	4259679	130
	MSR-CA	4498559	44
	SGA	5614693	2096
	SOAPdenovo	4627058	312
	Velvet	4615068	382

doi:10.1371/journal.pone.0155327.t003

numbers of gaps in most of the analyzed assemblies due the insert length, which was used to align against the reference sequences. In other cases, any gap that was closed had its length (the amount of Ns) increased, which occurred for the assemblies from SGA and SOAPdenovo for *S. aureus* and for the assemblies from SOAPdenovo for *R. sphaeroides*. Other results showed that GapFiller reduced the amount of gaps but increased their length (amount of Ns), which was observed for MSR-CA for *S. aureus* and CABOG, MSR-CA, SGA and for Velvet for *R. sphaeroides* (Table 5). Despite GapFiller having closed more gaps than GapBlaster for CABOG, MSR-CA, SGA and Velvet for *R. sphaeroides*, GapBlaster was superior to GapFiller. GapBlaster was able to fill more gaps and reduce the number of Ns in the sequences for nearly all GAGE assemblies, although it did not use paired reads.

FGAP filled more gaps than GapBlaster for all assemblies of the GAGE dataset. Nevertheless, GapBlaster filled more Ns than FGAP for ABySS and SGA for *S. aureus* and CABOG, MSR-CA and Velvet for *R. sphaeroides* (Table 5).

Despite FGAP performing the gap filling analysis automatically while GapBlaster performed the analysis manually, they achieved very similar results with respect to the number of gaps

Table 4. Gap closure results for the *Corynebacterium* genome.

	#Gaps	#N	#Gaps GB	#N GB	#Gaps FGAP	#N FGAP
Corynebacterium pseudotuberculosis	24	1794	11	931	5	360

Results of gap closure analysis of *Corynebacterium*, showing the #Gaps (amount of gaps) and #N (gap length); #Gaps GB and #N GB show the amount of remaining gaps and Ns, respectively, after the use of GapBlaster. The #Gaps FGAP and #N FGAP show the amount of remaining gaps and Ns, respectively, after the use of FGAP.

doi:10.1371/journal.pone.0155327.t004

Table 5. Gap closure results for GAGE Assemblies.

Staphylococcus aureus	**#Gaps**	**#N**	**#Gaps GB**	**#N GB**	**#Gaps FGAP**	**#N FGAP**	**#Gaps GF**	**#N GF**
AbySS	66	55882	55	47614	45	51127	69	56355
AbySS2	33	9391	27	7780	17	4850	35	10003
Allpaths-LG	23	9875	20	9446	15	8755	40	10472
Bambus2	95	29201	93	29159	80	27459	98	30771
MSR-CA	81	10353	72	7868	47	7861	80	11651
SGA	654	300607	642	292067	634	298252	654	312284
SOAPdenovo	9	4857	8	4837	7	4708	9	5010
Velvet	128	17688	124	17473	94	15406	127	19863
Rhodobacter sphaeroides	***#Gaps***	***#N***	***#Gaps GB***	***#N GB***	***#Gaps FGAP***	***#N FGAP***	***#Gaps GF***	***#N GF***
AbySS	261	114525	261	114525	256	113886	306	118298
AbySS2	235	62570	233	62128	228	60323	290	68052
Allpaths-LG	90	21329	87	20733	82	19500	164	24001
Bambus2	85	57041	83	56402	80	55990	84	56930
CABOG	193	21547	192	20892	190	21065	191	25011
MSR-CA	356	32628	349	26189	347	31174	336	37494
SGA	938	1145600	938	1145600	930	1144955	930	1159235
SOAPdenovo	38	10461	37	9601	37	10097	38	11176
Velvet	427	86815	424	86785	404	86063	415	94150

Results of the gap closure process for the data produced by GAGE with several assemblers for *S. aureus* and *R. sphaeroides*. Showing the #Gaps (amount of gaps) and #N (gap length); #Gaps GB and #N GB show the amount of remaining gaps and Ns, respectively, after the use of GapBlaster. The #Gaps FGAP and #N FGAP show the amount of remaining gaps and Ns, respectively, after the use of FGAP. The #Gaps GF and #N GF show the amount of remaining gaps and Ns, respectively, after the use of GapFiller.

doi:10.1371/journal.pone.0155327.t005

and N reductions for SOAPdenovo for *S. aureus* and Bambus2, CABOG, MSR-CA and SOAPdenovo for *R. sphaeroides* (Table 5).

FGAP showed better results for both the *C. pseudotuberculosis* and the GAGE datasets. We performed the gap filling analysis of the FGAP results with the original contigs of each

Table 6. Comparison of the original results of FGAP and after manual curation with GapBlaster.

Staphylococcus aureus	**#Gaps FGAP**	**#N FGAP**	**#Gaps after GB**	**#N after GB**
AbySS	45	51127	41	45439
MSR-CA	47	7861	46	6359
SGA	634	298252	629	290825
Rhodobacter sphaeroides	**#Gaps FGAP**	**#N FGAP**	**#Gaps after GB**	**#N after GB**
AbySS2	228	60323	227	60040
Allpaths-LG	82	19500	81	19494
Bambus2	80	55990	79	55402
CABOG	190	21065	188	19568
MSR-CA	347	31174	343	25592
SOAPdenovo	37	10097	36	9237
	#Gaps FGAP	**#N FGAP**	**#Gaps after GB**	**#N after GB**
Corynebacterium pseudotuberculosis	5	360	3	251

The results produced by FGAP were used as input for GapBlaster, and the organism/assemblies that were improved are shown. The #Gaps FGAP and #N FGAP show the amount of gaps and Ns, respectively, for the results of FGAP. The #Gaps **after** GB and #N **after** GB show the amounts of remaining gaps and Ns, respectively, after the use of GapBlaster.

doi:10.1371/journal.pone.0155327.t006

Table 7. Comparison of the original results of GapFiller and after manual curation with GapBlaster.

Staphylococcus aureus	**#Gaps GF**	**#N GF**	**#Gaps after GB**	**#N after GB**
AbySS	69	56355	66	54837
AbySS2	35	10003	30	8741
Allpaths-LG	40	10472	39	10455
Bambus2	98	30771	97	30725
MSR-CA	80	11651	76	9794
SGA	654	312284	646	307095
Rhodobacter sphaeroides	**#Gaps GF**	**#N GF**	**#Gaps after GB**	**#N after GB**
AbySS	306	118298	304	118287
AbySS2	290	68052	288	67740
Allpaths-LG	164	24001	163	23780
CABOG	191	25011	190	24336
MSR-CA	336	37494	333	33590
SGA	930	1159235	929	1159162

The results produced by GapFiller were used as input for GapBlaster, and the organism/assemblies that were improved are shown. The #Gaps GF and #N GF show the amount of gaps and Ns, respectively, in the results of GapFiller. The #Gaps **after** GB and #N **after** GB show the amount of remaining gaps and Ns, respectively, after the use of GapBlaster.

doi:10.1371/journal.pone.0155327.t007

organism and assembly through GapBlaster to determine whether GapBlaster could improve the results produced by FGAP. The results are shown in Table 6. Compared with the FGAP results, GapBlaster improved 55.55% of all assemblies of the GAGE dataset and *C. pseudotuberculosis*. GapFiller was not used for this comparison of *Corynebacterium* data because only a fragment library was available for this organism.

GapBlaster improved the results of FGAP for *C. pseudotuberculosis* in that it reduced the number of gaps from 5 to 3. Therefore, GapBlaster improved the gap filling results for several assemblies for *S. aureus* and *R. sphaeroides*, as shown in Table 6. This analysis shows that despite its usefulness for closing gaps through its GUI, GapBlaster is also useful for gap filling when used in combination with another tools.

Similar to the analysis of the FGAP results, we conducted an evaluation of the GapFiller output files and the original contigs of each organism/assembler of the GAGE dataset via GapBlaster. Compared with the GapFiller results, GapBlaster improved 70.58% of all assemblies of the GAGE dataset (Table 7).

GapBlaster improved the results of GapFiller for almost all of the CAGE data (Table 7). The best gap filling results were ABySS2 and SGA for *S. aureus*, where the gaps decreased from 35 to 30 and 654 to 646, respectively (Table 7). Beyond being a very useful tool with an interface for manual curation, GapBlaster is a valuable open source program that can be used with other tools in the gap filling analysis to produce more complete genome drafts.

To evaluate the accuracy of the closed gaps, all results produced by GapBlaster, FGAP, GapFiller and the original files (scaffolds) were aligned against their respective genome reference (Table 2). The results show that all of the files produced in the gap filling analysis showed similar alignment percentages with the original files, which confirms that the bases introduced in the filled gaps were correct (S1 Table).

Despite the three methods (Blast+, Blast Legacy, and Mummer) implemented in GapBlaster, we used only Blast+ to fill gaps as this method is the same used for FGAP software. However, we tested all of the algorithms for GAGE data, and Blast Legacy and Blast+ presented similar results (S2 Table). The comparisons of the features of GapBlaster, FGAP and GapFiller helped

Table 8. Comparison of the features of GapBlaster, FGAP and GapFiller.

Features Alignment method	GapBlaster Blast+ or Blast Legacy or Mummer	FGAP Blast+	GapFiller Bowtie or BWA
Set Flank Alignment	Yes	Yes	Yes
Allow Manual Curation	Yes	No	No
Perform Automatic Analysis	Yes	Yes	Yes
Based on paired-reads	No	No	Yes
Use contigs to fill gaps	Yes	Yes	No
Graphical interface	Yes	No	No
Improve gap filling results of other softwares	Yes	Not tested	Not tested
Correctly fill gaps?	Yes	Yes	Yes

doi:10.1371/journal.pone.0155327.t008

to identify the main advantage of GapBlaster, the graphical interface, which uses contigs to fill gaps and allows manual curation (Table 8).

Conclusions

Despite the efficiency of tools such as FGAP and GapFiller, the gap closure process continues to be a step that requires manual curation for the acquisition of high quality results, such as those presented by GapBlaster, the use of which is simplified by the graphical interface.

GapBlaster revealed improved gap filling performance using contigs compared to GapFiller for nearly all data evaluated despite the use of paired reads in GapFiller.

In addition to presenting better results, the GapBlaster program has the advantage of introducing fewer errors, based on the ability of the interface to allow the user to decide if a gap is filled properly. As an alternative, GapBlaster can be used in addition to other gap closer programs to facilitate genome completion through manual manipulation, as was shown in the analysis of the GapBlaster program to improve the results of FGAP and GapFiller.

Supporting Information

S1 Fig. GapBlaster main interface. The main graphical interface through which the user can input the contigs and scaffold files and set the alignment preferences.
(TIF)

S2 Fig. Alignment interface. The screen shows the results of the alignment of a contig against a scaffold. All alignments produced are listed. The user can check if the alignments are correct and select them.
(TIF)

S3 Fig. Selected Alignment. The aligned contig filled the gap with high accuracy due to the high identity found in the gap flanks.
(TIF)

S1 Table. GapBlaster, FGAP and GapFiller accuracy. This table shows information about the percentage of bases aligned against the GapBlaster, FGAP, GapFiller results and the original scaffolds to the reference genome to evaluate if the filled gaps introduced the correct bases in the analysis.
(XLS)

S2 Table. GapBlaster algorithm comparison. Comparison of the three alignment algorithms implemented in GapBlaster (Blast+, Blast Legacy, Mummer) to evaluate the number of

alignments identified, closed gaps and N removed after the gap filling process.
(XLS)

Acknowledgments

This work was part of the Rede Paraense de Genômica e Proteômica and was supported by Fundação de Amparo a Pesquisa do Estado do Pará. Conselho Nacional de Desenvolvimento Científico e Tecnológico (CNPq).

Author Contributions

Conceived and designed the experiments: PHCGS FM RTJR. Performed the experiments: PHCGS FM KP RTJR. Analyzed the data: PHCGS AV DMM SS KP LG RTJR. Contributed reagents/materials/analysis tools: RTJR AS VA. Wrote the paper: PHCGS AV DMM SS KP LG VA AS RTJR.

References

1. Liu L, Li Y, Li S, Hu N, He Y, Pong R, et al. Comparison of next-generation sequencing systems. J Biomed Biotechnol. 2012; 2012: 251364. doi: 10.1155/2012/251364 PMID: 22829749
2. Wojcieszek M, Pawełkowicz M, Nowak R, Przybecki Z. Genomes correction and assembling: present methods and tools. SPIE Proc. 2014; 9290: 92901X. doi: 10.1117/12.2075624
3. Luo R, Liu B, Xie Y, Li Z, Huang W, Yuan J, et al. SOAPdenovo2: an empirically improved memory-efficient short-read de novo assembler. 2012; 1–6.
4. Zerbino DR, Birney E. Velvet: algorithms for de novo short read assembly using de Bruijn graphs. Genome Res. 2008; 18: 821–9. doi: 10.1101/gr.074492.107 PMID: 18349386
5. Boetzer M, Henkel C V, Jansen HJ, Butler D, Pirovano W. Scaffolding pre-assembled contigs using SSPACE. Bioinformatics. 2011; 27: 578–9. doi: 10.1093/bioinformatics/btq683 PMID: 21149342
6. Boetzer M, Pirovano W. Toward almost closed genomes with GapFiller. Genome Biol. BioMed Central Ltd; 2012; 13: R56. doi: 10.1186/gb-2012-13-6-r56 PMID: 22731987
7. Ekblom R, Wolf JBW. A field guide to whole-genome sequencing, assembly and annotation. Evol Appl. 2014; 7: 1026–1042. doi: 10.1111/eva.12178 PMID: 25553065
8. Ramos RTJ, Carneiro AR, Caracciolo PH, Azevedo V, Schneider MPC, Barh D, et al. Graphical contig analyzer for all sequencing platforms (G4ALL): a new stand-alone tool for finishing and draft generation of bacterial genomes. Bioinformation. 2013; 9: 599–604. doi: 10.6026/97320630009599 PMID: 23888102
9. Piro V, Faoro H, Weiss V. FGAP: an automated gap closing tool. BMC Res Notes. 2014; 1–5. doi: 10.1186/1756-0500-7-371 PMID: 24938749
10. Paulino D, Warren RL, Vandervalk BP, Raymond A, Jackman SD, Birol I. Sealer: a scalable gap-closing application for finishing draft genomes. BMC Bioinformatics. BMC Bioinformatics; 2015; 16: 230. doi: 10.1186/s12859-015-0663-4 PMID: 26209068
11. Touchman J. Comparative Genomics. In: Comparative Genomics. Nature Education Knowledge 3 (10):13 [Internet]. 2010. Available: http://www.nature.com/scitable/knowledge/library/comparative-genomics-13239404
12. Seib KL, Zhao X, Rappuoli R. Developing vaccines in the era of genomics: a decade of reverse vaccinology. Clin Microbiol Infect. 2012; 18: 109–116. doi: 10.1111/j.1469-0691.2012.03939.x PMID: 22882709
13. Theodore MJ, Anderson RD, Wang X, Katz LS, Vuong JT, Bell ME, et al. Evaluation of new biomarker genes for differentiating Haemophilus influenzae from Haemophilus haemolyticus. J Clin Microbiol. 2012; 50: 1422–1424. doi: 10.1128/JCM.06702-11 PMID: 22301020
14. Tatti KM, Loparev VN, Ranganathanganakammal S, Changayil S, Frace M, Weil MR, et al. Draft genome sequences of Bordetella holmesii strains from blood (F627) and nasopharynx (H558). Genome Announc. 2013; 1: e0005613. doi: 10.1128/genomeA.00056-13 PMID: 23516195
15. Mira A, Martín-Cuadrado A. The bacterial pan-genome: a new paradigm in microbiology. Int Microbiol. 2010; 45–57. doi: 10.2436/20.1501.01.110 PMID: 20890839
16. Altschul SF, Gish W, Miller W, Myers EW, Lipman DJ. Basic local alignment search tool. J Mol Biol. 1990; 215: 403–410. doi: 10.1016/S0022-2836(05)80360-2 PMID: 2231712

17. Delcher A, Kasif S. Alignment of whole genomes. Nucleic Acids Res. 1999; 27: 2369–2376. Available: http://nar.oxfordjournals.org/content/27/11/2369.short PMID: 10325427

18. Salzberg SL, Phillippy AM, Zimin A, Puiu D, Magoc T, Koren S, et al. GAGE: A critical evaluation of genome assemblies and assembly algorithms. Genome Res. 2012; 22: 557–67. doi: 10.1101/gr.131383.111 PMID: 22147368

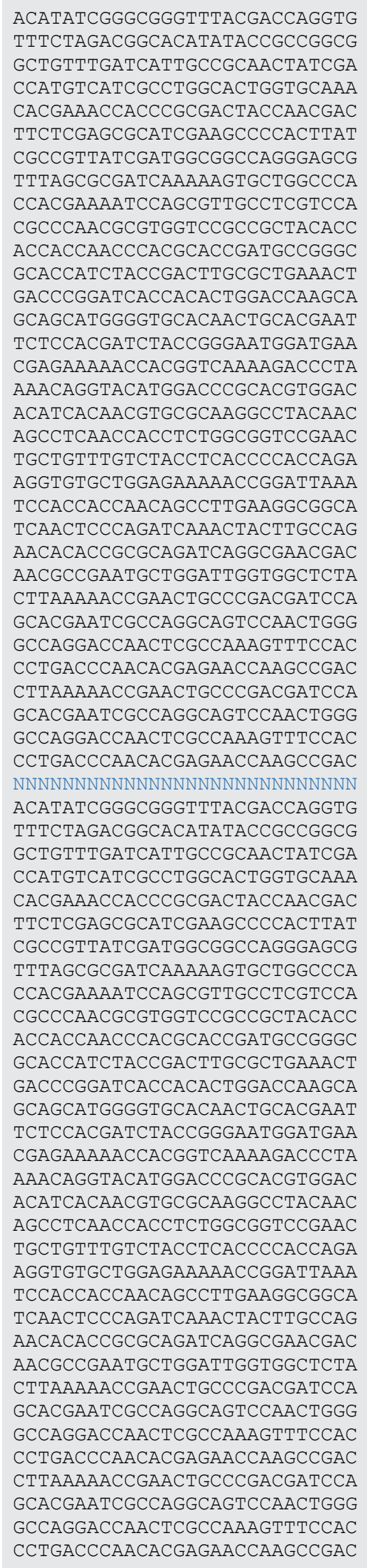
19. Dorella FA, Pacheco LG, Oliveira SC, Miyoshi A, Azevedo V. Corynebacterium pseudotuberculosis: microbiology, biochemical properties, pathogenesis and molecular studies of virulence. Vet Res. 2006; 37: 201–218. doi: 10.1051/vetres PMID: 16472520

20. Bankevich A, Nurk S, Antipov D, Gurevich A a., Dvorkin M, Kulikov AS, et al. SPAdes: A New Genome Assembly Algorithm and Its Applications to Single-Cell Sequencing. J Comput Biol. 2012; 19: 455–477. doi: 10.1089/cmb.2012.0021 PMID: 22506599

ANEXO B – MANUAL DO USUÁRIO

GapBlaster User Guide v1.0.3

Fábio M. Miranda
Federal University of Pará



Preface

This user guide is intended to provide a concise reference on how to install and operate GapBlaster, and it is distributed in the hope that it will be useful in providing you with the best user experience.

Copyright Notice

Acknowledgments

We would like to thank the Brazilian National Council for Scientific and Technological Development (CNPq), the Coordination for the Improvement of Higher Education Personnel (CAPES) and the Genomics and Bioinformatics Laboratory of the Federal University of Pará, for supporting the development of this project.

The author also would like to thank PhD Rommel Thiago Juca Ramos, from the Federal University of Pará, for his supervision.

Contact Information

Please address errors in this documentation, questions, suggestions or bugs in the software to miranda.fmm@gmail.com. Your feedback is highly appreciated and will help us improve the quality of our work, in order to better serve you and other users.

Table of Contents

1. Overview

GapBlaster is a software that aims at closing scaffolds' gaps quickly and without much user's prior knowledge in bioinformatics. Its graphical user interface was developed in a way that even users without much computer knowledge should be able to use it. In the next sections, we explain the requirements to run GapBlaster, how to install dependencies and how to use it.

2. System Requirements

GapBlaster is written in Java and is compiled with JDK 8. Therefore, in order to be able to run it, Java 8 or higher needs to be installed. For more information or installation instructions, please visit: http://java.com/en/download/manual.jsp. Alternatively, you can install OpenJDK. More information can be found at http://openjdk.java.net/.

GapBlaster also requires some third party software to run successfully. If these utilities are absent, GapBlaster may fail to run correctly. The next section explains how to obtain and install them.

3. Installing Dependencies on Debian

GapBlaster does not require installation. However, it relies on third party software to perform local alignments, i.e., Legacy Blast, MUMmer or Blast+. It is not necessary to install all of them, only the ones you are going to use. The following subsections explain how to install the local aligner of your choice on Debian:

3.1 Installing Legacy Blast

To install Legacy Blast on Debian, run the following command in a terminal as root:

```
sudo apt-get install blast2
```

3.2 Installing MUMmer

To install MUMmer on Debian, run the following command in a terminal as root:

```
sudo apt-get install mummer
```

3.3 Installing Blast+

To install Blast+ on Debian, run the following command in a terminal as root:

```
sudo apt-get install ncbi-blast+
```

4. Running GapBlaster

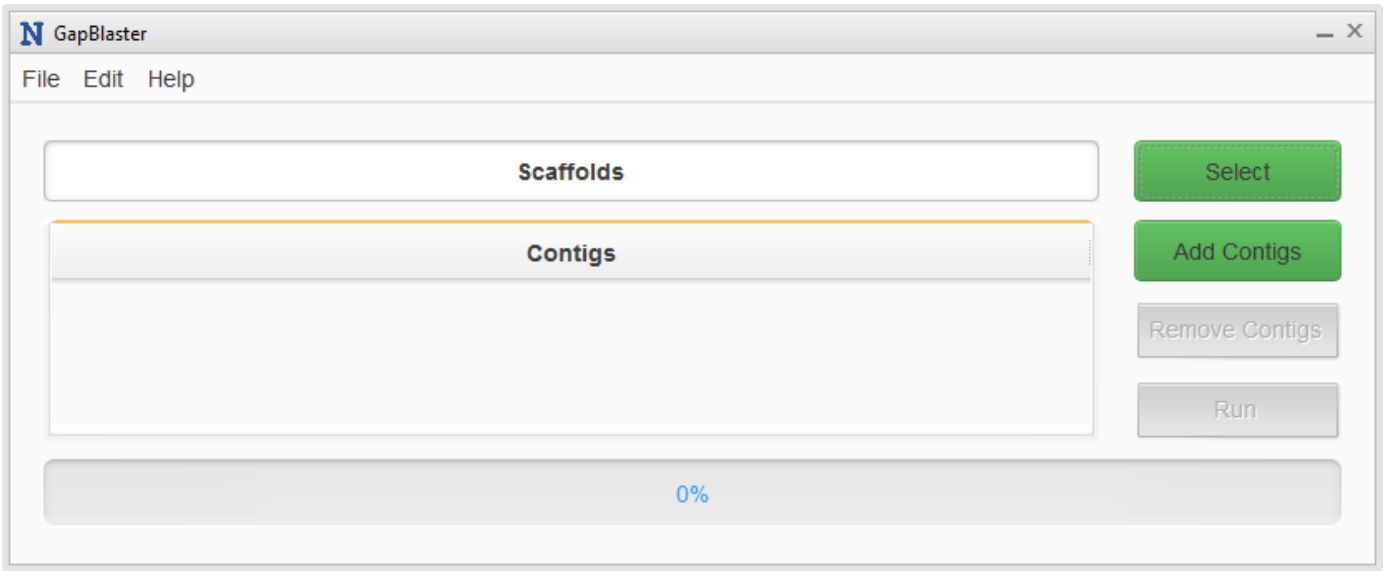
To start GapBlaster, run the following command in a terminal in the same folder where you downloaded the jar file:

```
java –jar GapBlaster _v1.1.2.jar
```

You can allocate more memory for GapBlaster if necessary. For example, if you want to allocate a minimum of 5GB and a maximum of 10GB for GapBlaster, add the following parameters to the previous command:

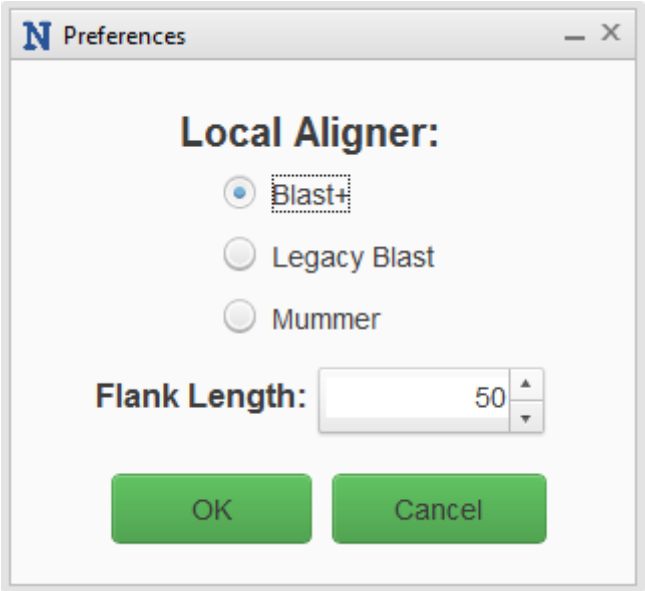
```
java –Xmx10g –Xms5g –jar GapBlaster _v1.1.2.jar
```

Once GapBlaster starts, you should be able to see the following window:



4.1 Editing the Preferences

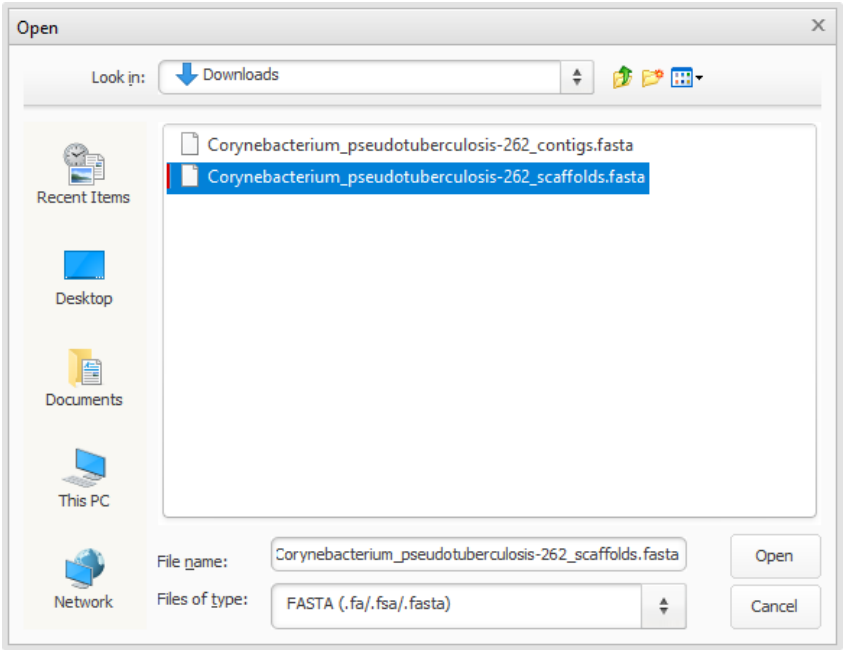
In the preferences window you can choose the local aligner and set the minimum flank length to perform alignments. To edit the preferences, click “edit → preferences” or press “Ctrl + P” in the main screen. This action will open the following window:



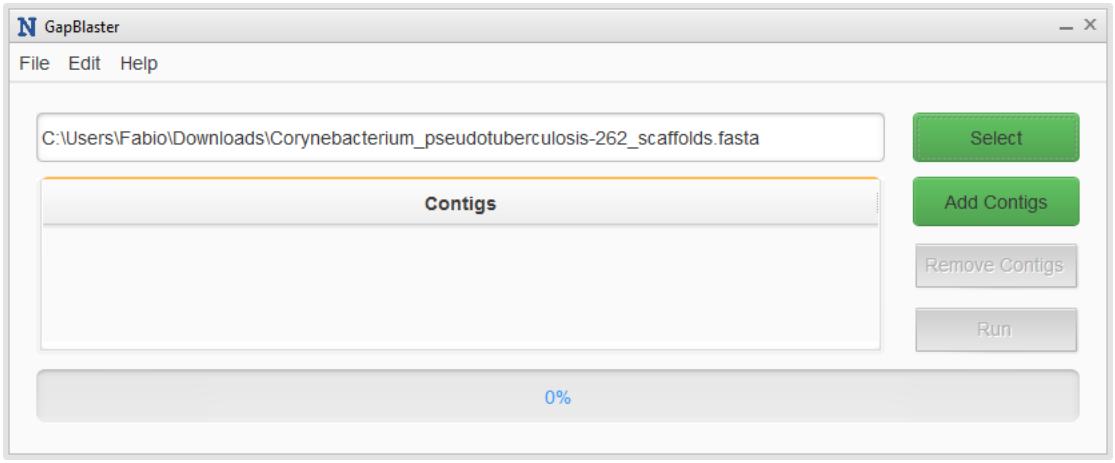
When you finish setting the preferences, click the OK button to save the changes.

4.2 Selecting the Input Files

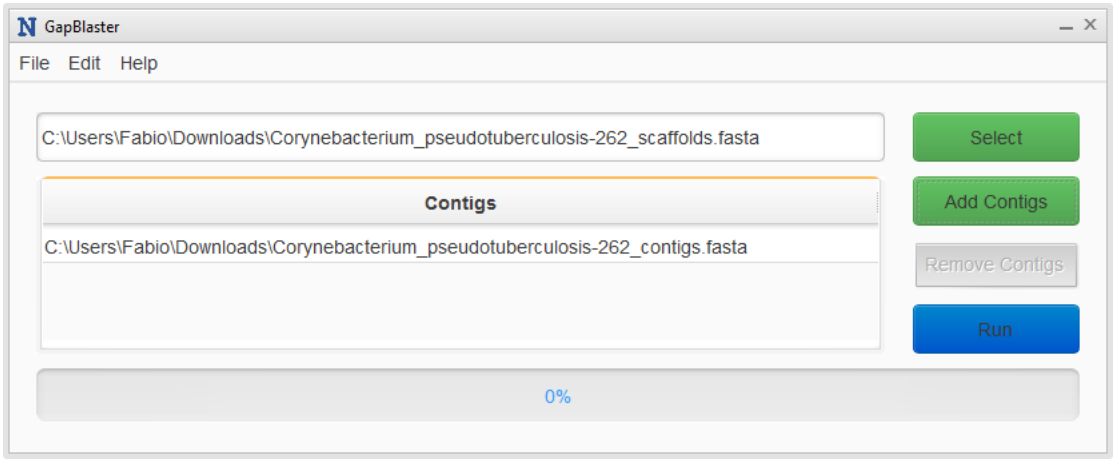
To select the scaffolds file, simply click the “select” button on the main screen, and a dialog box will pop up.



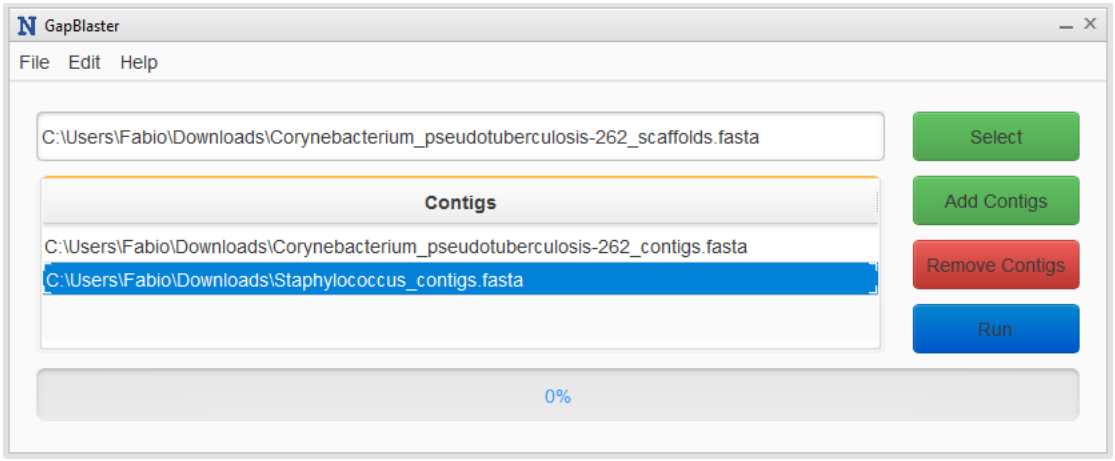
In this dialog box, you can choose your file. Once you finish choosing, click the open button and the path of the scaffolds file will be shown in the main screen.



To add one or more contigs files, click the button “Add Contigs” in the main screen and another dialog box will pop up. After selecting the files, it should look like this:



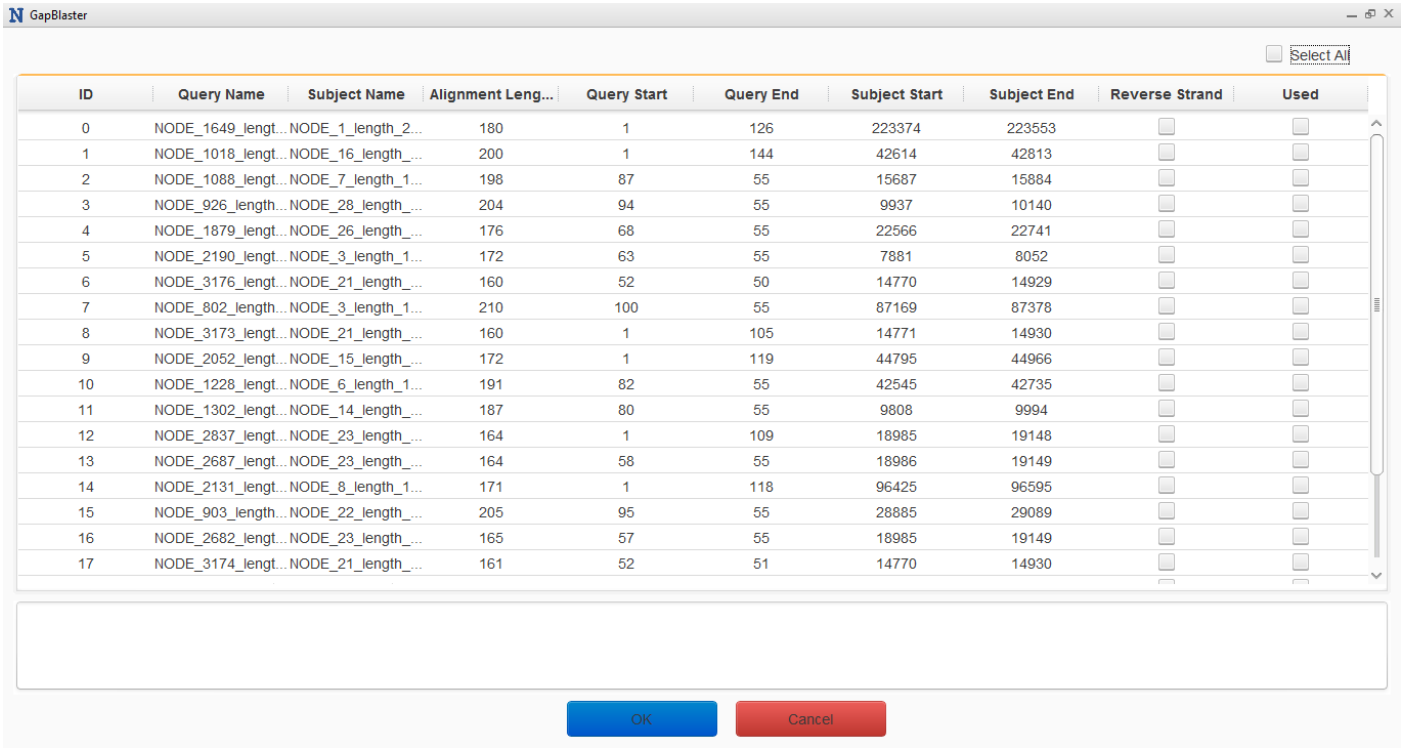
Notice that if you click on the path of a contigs file, the button remove contigs will be enabled. This way you can remove files that were added by mistake.



After you finish selecting all the files, click the “Run” button to perform the alignments.

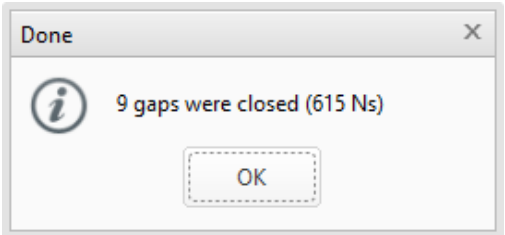
4.3 Selecting alignments

After you click the “Run” button, GapBlaster may take some time to perform all the steps required. When the program finishes performing the alignments, the following window will appear:



This window is where you can inspect the alignments made by GapBlaster. You can click on each one of them and choose whether that gap should be closed, by marking the box “Used” on the last column of an alignment. However, if you are confident that you want to apply all the modifications suggested by GapBlaster, you can simply click the button “Select All” on the top right of the screen.

When you finish selecting all the gaps that you want to be closed, click the “OK” button on the bottom of the screen. GapBlaster will apply all the changes that you selected in the scaffolds and print how many gaps were closed. In the end you should see a window like this:

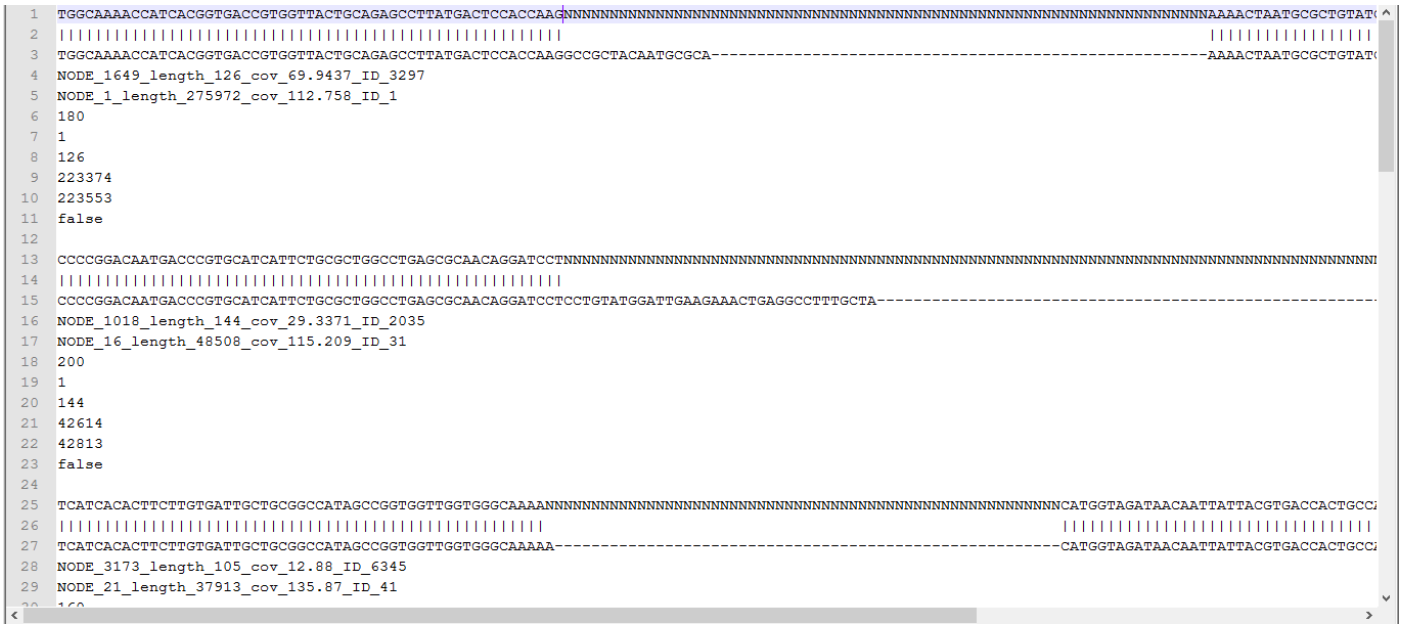


4.4 Output Files

GapBlaster generates two files as output: the new scaffolds file with the closed gaps (it has an extension "*_gapblasted.fasta") and a log file with the selected alignments.

Here is an example of what the original scaffolds file looks like (left picture) and what GapBlaster changed in it (right picture):

The log file contains the information regarding the alignments that were performed to close the gaps. It looks like this:



Each set contains the following information for an alignment:

- Subject Sequence
- Middle Sequence
- Query Sequence
- Query Name
- Subject Name
- Alignment Length
- Query Start
- Query End
- Subject Start
- Subject End
- Reverse Strand (true or false)