UNIVERSIDADE TECNOLÓGICA FEDERAL DO PARANÁ

BRENO CESAR DUPIN

TIGRE: UMA FERRAMENTA ESCALÁVEL PARA EXTRAÇÃO DE REGIÕES INTERGÊNICAS A PARTIR DE ANOTAÇÕES GFF3

CORNÉLIO PROCÓPIO

2025

versão 11.0 (abr. 25)

BRENO CESAR DUPIN

TIGRE: UMA FERRAMENTA ESCALÁVEL PARA EXTRAÇÃO DE REGIÕES INTERGÊNICAS A PARTIR DE ANOTAÇÕES GFF3

TIGRE: Tool for InterGenic Region Extraction

Trabalho de Conclusão de Curso de Graduação apresentado (a) como requisito para obtenção do título de Bacharel em Engenharia da Computação da Universidade Tecnológica Federal do Paraná.
Orientador(a): Profª. Drª. Tatianne Costa Negri Rocha

CORNÉLIO PROCÓPIO

2025

BRENO CESAR DUPIN

TIGRE: UMA FERRAMENTA ESCALÁVEL PARA EXTRAÇÃO DE REGIÕES INTERGÊNICAS A PARTIR DE ANOTAÇÕES GFF3

Trabalho de Conclusão de Curso de Graduação apresentado (a) como requisito para obtenção do título de Bacharel em Engenharia da Computação da Universidade Tecnológica Federal do Paraná.

Data de aprovação: 26 / novembro / 2026

Tatianne Costa Negri Rocha
Doutorado
Universidade Tecnológica Federal do Paraná

Glaucia Maria Bressan
Doutorado
Universidade Tecnológica Federal do Paraná

Andre Yoshiaki Kashiwabara
Doutorado
Universidade Federal de Sergipe

CORNÉLIO PROCÓPIO

2025

RESUMO

A extração de regiões intergênicas a partir de arquivos de anotação genômica é essencial para estudos de genômica comparativa e biologia evolutiva, porém apresenta desafios computacionais ao lidar com *features* sobrepostas, genomas circulares e processamento em larga escala. O objetivo geral deste trabalho é apresentar a TIGRE (*Tool for Intergenic Regions Extraction*), uma ferramenta de linha de comando desenvolvida em *Python* para extração automatizada de regiões intergênicas de arquivos GFF3. Originalmente concebida como etapa de pré-processamento em um projeto de pesquisa sobre organelas de plantas, a ferramenta foi generalizada e transformada em uma solução *standalone*, encontrando-se atualmente em processo de publicação na revista *STAR Protocols*. A TIGRE implementa três comandos sequenciais: *clean*, para preparação e padronização de arquivos GFF3, incluindo resolução de sobreposições e fragmentação de features que atravessam limites de genomas circulares; *extract*, para identificação e anotação de regiões intergênicas; e *getfasta*, para recuperação de sequências de nucleotídeos. A arquitetura permite processamento tanto de genomas individuais quanto em lote, com suporte a paralelização via multiprocessamento e otimização de memória através de arquitetura servidor-cliente quando integrada com a ferramenta GDT. A validação foi realizada através do processamento de 1.207 genomas mitocondriais de plantas, demonstrando execução eficiente com tempo total de 11.1 segundos em servidor e consumo máximo de memória de 155.7 MB. A ferramenta está disponível como *software* de código aberto sob licença MIT, distribuída via PyPI, e já processou mais de 35.000 genomas em seu uso contínuo no projeto de pesquisa original. A TIGRE representa uma contribuição significativa para a comunidade de bioinformática, oferecendo solução robusta, escalável e flexível para extração de regiões intergênicas, facilitando análises genômicas comparativas em larga escala.

Palavras-chave: bioinformática; genômica; regiões intergênicas.

ABSTRACT

The extraction of intergenic regions from genomic annotation files is essential for comparative genomics and evolutionary biology studies, but presents computational challenges when dealing with overlapping features, circular genomes, and large-scale processing. The overall objective of this work is to present TIGRE (Tool for Intergenic Regions Extraction), a command-line tool developed in Python for automated extraction of intergenic regions from GFF3 files. Originally conceived as a preprocessing step in a research project on plant organelles, the tool was generalized and transformed into a standalone solution, currently undergoing publication in STAR Protocols journal. TIGRE implements a pipeline of three sequential commands: clean, for preparation and standardization of GFF3 files, including overlap resolution and fragmentation of features spanning circular genome boundaries; extract, for identification and annotation of intergenic regions; and getfasta, for retrieval of nucleotide sequences. The architecture supports both single-genome and batch processing, with parallelization via multiprocessing and memory optimization through server-client architecture when integrated with the GDT tool. Validation was performed through processing of 1,207 plant mitochondrial genomes, demonstrating efficient execution with total time of 11.1 seconds on server and maximum memory consumption of 155.7 MB. The tool is available as open-source software under MIT license, distributed via PyPI, and has already processed over 35,000 genomes in its continued use in the original research project. TIGRE represents a significant contribution to the bioinformatics community, offering a robust, scalable, and flexible solution for intergenic region extraction, facilitating large-scale comparative genomic analyses.

Keywords: bioinformatics; genomics; intergenic regions.

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE SÍMBOLOS

TIGRE	Tool for InterGenic Region Extraction
CLI	Command Line Interface
GDT	Gene Dictonary Tool
NCBI	National Center for Biotechnology Information
GFF3	General Feature Format, version 3
CDS	coding sequences
AN	NCBI Accession Number
PyPI	Python Package Index

SUMÁRIO

1 INTRODUÇÃO

Este trabalho apresenta a TIGRE (*Tool for Intergenic Regions Extraction*), uma ferramenta de linha de comando (CLI - *Command Line Interface*) para extração e processamento automatizado de regiões intergênicas a partir de genomas anotados. Desenvolvida para suprir uma necessidade específica na área de bioinformática, a ferramenta possibilita a extração eficiente dessas regiões utilizando arquivos de anotação genômica (GFF3) como entrada, facilitando estudos comparativos e análises evolutivas em larga escala.

TIGRE foi desenvolvida como parte de um projeto de pesquisa em colaboração com o doutorando Matheus Sanita Lima (Departamento de Biologia, *Western University*, Canadá), focado na análise de regiões intergênicas de organelas de plantas. Durante o desenvolvimento desse projeto, identificou-se que determinadas etapas do de préprocessamento de análise, especificamente a padronização de anotações genômicas e a extração de regiões intergênicas, poderiam ser generalizadas e disponibilizadas como ferramentas independentes, beneficiando a comunidade científica mais ampla. Esse processo resultou no desenvolvimento de duas ferramentas: a GDT - *Gene Dictionary Tool* (Dupin *et al.*, 2025) -, para padronização na nomenclatura dos elementos genômicos, e a TIGRE, para extração de regiões intergênicas. Ambas possuem manuscritos de protocolo (*method papers*) que foram aceitos para publicação na revista *STAR Protocols*.

A motivação técnica para o desenvolvimento da ferramenta surgiu durante o projeto realizado em colaboração com o doutorando Matheus Sanita Lima, quando se constatou que o comando *complement* do pacote BEDTools (Quinlan; Hall, 2010) apresentava limitações que impediam sua aplicação direta às necessidades da pesquisa. Além de não oferecer suporte adequado a genomas circulares, estrutura típica de organelas, como cloroplastos e mitocôndrias, o *bedtools complement* não preserva informações sobre as features que delimitam cada intervalo resultante, comprometendo análises que dependem da relação entre elementos adjacentes. Somado a isso, o uso de anotações provenientes do NCBI (*National Center for Biotechnology Information* RefSeq (Goldfarb *et al.*, 2024) introduzia uma dificuldade adicional: os arquivos GFF incluem uma variedade de features além de genes, o que exigia etapas prévias de filtragem e padronização antes que o BEDTools pudesse ser aplicado corretamente. TIGRE foi desenvolvida precisamente para superar essas limitações, incorporando suporte nativo a genomas circulares e mantendo de forma explícita as anotações que definem cada região intergênica.

Nesse contexto, o objetivo geral deste trabalho é apresentar a TIGRE como uma ferramenta capaz de extrair, padronizar e caracterizar o contexto das regiões intergênicas de forma automatizada, incluindo suporte pleno a genomas circulares.

1.1 Contextualização Geral

Os avanços nas tecnologias de sequenciamento genômico têm gerado um volume crescente de dados biológicos, demandando ferramentas computacionais eficientes para análise e

interpretação (León; Pastor, 2021). Um genoma sequenciado consiste em uma longa sequência de nucleotídeos (A, T, C e G) que, para ser cientificamente útil, precisa passar por anotação genômica - processo que identifica e cataloga regiões funcionais como genes e RNAs (Guigó, 2023). Essas anotações funcionam como um “mapa” do genoma e são armazenadas em arquivos de formatos padronizados (como GFF3), permitindo processamento consistente por diferentes ferramentas.

Em muitos organismos, a maior parte do genoma (nuclear e organelar) corresponde a regiões intergênicas, segmentos localizados entre os genes. Embora sua função biológica seja objeto de debate científico (Eddy, 2013), o estudo dessas regiões tem se mostrado cada vez mais relevante para a compreensão da regulação gênica e da evolução genômica. Regiões intergênicas contêm elementos regulatórios essenciais, como promotores, *enhancers* e silenciadores, que controlam a expressão dos genes adjacentes (Nelson; Hersh; Carroll, 2004). Além disso, análises comparativas dessas regiões entre diferentes espécies têm revelado padrões de conservação evolutiva que sugerem funções biológicas ainda não completamente elucidadas, incluindo a produção de RNAs não codificantes e a manutenção da estrutura tridimensional da cromatina (Mattick *et al.*, 2023). A maioria dos estudos sobre regiões intergênicas foi realizada em genomas nucleares. Por conseguinte, o estudo das regiões integênicas de genomas de organelas é um nicho inexplorado (Sanita Lima *et al.*, 2024).

Essas regiões representam um conjunto de dados desafiador para extração e análise automatizada em larga escala. O desenvolvimento de ferramentas computacionais capazes de processar eficientemente essas regiões é fundamental para viabilizar estudos evolutivos e comparativos entre diferentes organismos.

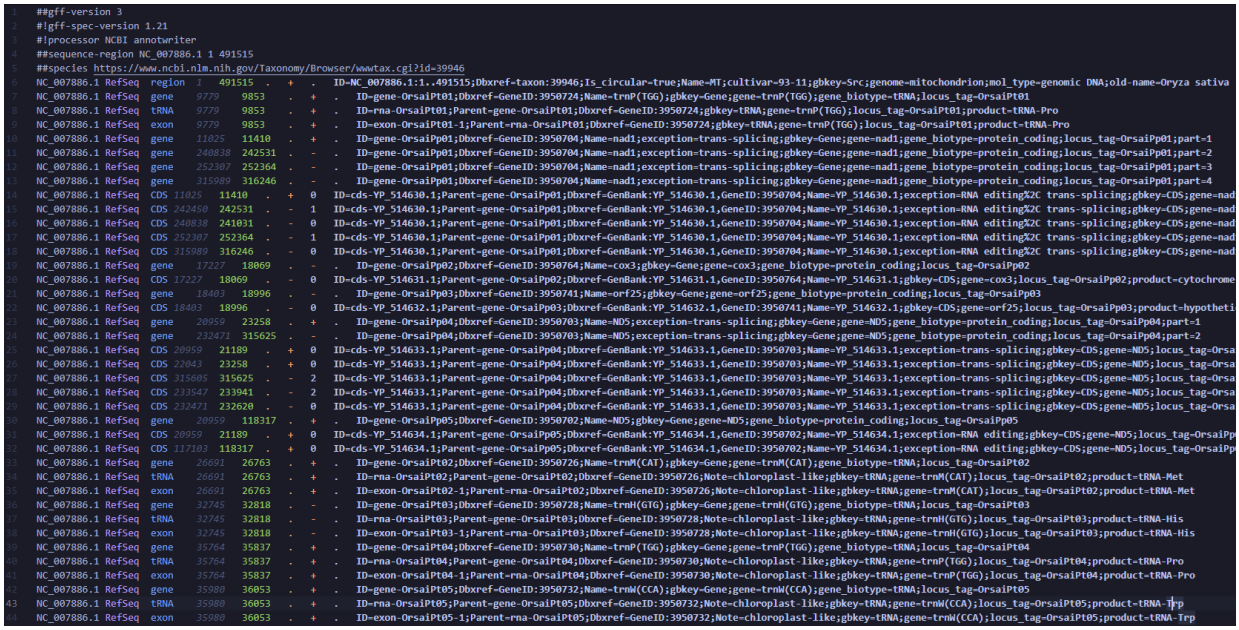
Para possibilitar essa extração automatizada, é essencial compreender como as anotações genômicas são estruturadas e armazenadas. O formato mais amplamente utilizado para esse propósito é o GFF3 - *General Feature Format, version 3*, (The Sequence Ontology, 2024) -, um padrão de arquivo texto que organiza informações sobre as diferentes regiões funcionais do genoma de forma consistente e processável computacionalmente.

1.2 O Formato GFF3

O GFF3 é um formato de arquivo texto padronizado para representar anotações genômicas. No contexto de anotações genômicas, ***features*** são elementos ou regiões de interesse identificadas no genoma, podendo ser genes, éxons, RNAs, regiões regulatórias, entre outros. Estruturado como uma tabela com nove colunas separadas por tabulações, cada linha do GFF3 representa uma dessas ***features***.

A Figura 1 apresenta um exemplo de arquivo GFF3 do genoma mitocondrial da *Oryza sativa* (arroz), ilustrando a estrutura típica desse formato. Linhas iniciadas com ’#’ são comentários contendo metadados do arquivo, como a versão do formato e informações sobre a sequência. As colunas são: (1) *seqid* - identificador da sequência (NC_007886.1 no exemplo), (2) *source* - origem da anotação (RefSeq), (3) *type* - tipo da *feature* (gene, exon, CDS, tRNA, etc.), (4)

Figura 1 – GFF3 do genoma mitocondrial de *Oryza sativa* (NC_007886.1)



```
##gff-version 3
#!gff-spec-version 1.21
#!processor NCBI annotwriter
##sequence-region NC_007886.1 1 491515
##species https://www.ncbi.nlm.nih.gov/Taxonomy/Browser/wwwtax.cgi?id=39946
NC_007886.1 RefSeq  region   1      491515  .  +  .  ID=NC_007886.1:1..491515;Dbxref=taxon:39946;Is_circular=true;Name=MT;cultivar=93-11;gbkey=Src;genome=mitochondrion;mol_type=genomic DNA;old-name=Oryza sativa
NC_007886.1 RefSeq  gene     9779   9853    .  +  .     ID=gene-OrsaPt01;Dbxref=GeneID:3950724;Name=trnP(TGG);gbkey=Gene;gene=trnP(TGG);gene_biotype=tRNA;locus_tag=OrsaPt01
NC_007886.1 RefSeq  tRNA     9779   9853    .  +  .     ID=rna-OrsaPt01;Parent=gene-OrsaPt01;Dbxref=GeneID:3950724;gbkey=tRNA;gene=trnP(TGG);locus_tag=OrsaPt01;product=tRNA-Pro
NC_007886.1 RefSeq  exon     9779   9853    .  +  .     ID=exon-OrsaPt01-1;Parent=rna-OrsaPt01;Dbxref=GeneID:3950724;gbkey=tRNA;gene=trnP(TGG);locus_tag=OrsaPt01;product=tRNA-Pro
NC_007886.1 RefSeq  gene     11025  11410   .  +  .     ID=gene-OrsaPp01;Dbxref=GeneID:3950704;Name=nad1;exception=trans-splicing;gbkey=Gene;gene=nad1;gene_biotype=protein_coding;locus_tag=OrsaPp01;part=1
NC_007886.1 RefSeq  gene     240838 242531  .  -  .     ID=gene-OrsaPp01;Dbxref=GeneID:3950704;Name=nad1;exception=trans-splicing;gbkey=Gene;gene=nad1;gene_biotype=protein_coding;locus_tag=OrsaPp01;part=2
NC_007886.1 RefSeq  gene     252307 252364  .  -  .     ID=gene-OrsaPp01;Dbxref=GeneID:3950704;Name=nad1;exception=trans-splicing;gbkey=Gene;gene=nad1;gene_biotype=protein_coding;locus_tag=OrsaPp01;part=3
NC_007886.1 RefSeq  gene     315989 316246  .  -  .     ID=gene-OrsaPp01;Dbxref=GeneID:3950704;Name=nad1;exception=trans-splicing;gbkey=Gene;gene=nad1;gene_biotype=protein_coding;locus_tag=OrsaPp01;part=4
NC_007886.1 RefSeq  CDS  11025  11410   .  +  0  ID=cds-YP_514630.1;Parent=gene-OrsaPp01;Dbxref=GenBank:YP_514630.1,GeneID:3950704;Name=YP_514630.1;exception=RNA editing%2C trans-splicing;gbkey=CDS;gene=nad
NC_007886.1 RefSeq  CDS  242450 242531  .  -  1  ID=cds-YP_514630.1;Parent=gene-OrsaPp01;Dbxref=GenBank:YP_514630.1,GeneID:3950704;Name=YP_514630.1;exception=RNA editing%2C trans-splicing;gbkey=CDS;gene=nad
NC_007886.1 RefSeq  CDS  240838 241031  .  -  0  ID=cds-YP_514630.1;Parent=gene-OrsaPp01;Dbxref=GenBank:YP_514630.1,GeneID:3950704;Name=YP_514630.1;exception=RNA editing%2C trans-splicing;gbkey=CDS;gene=nad
NC_007886.1 RefSeq  CDS  252307 252364  .  -  1  ID=cds-YP_514630.1;Parent=gene-OrsaPp01;Dbxref=GenBank:YP_514630.1,GeneID:3950704;Name=YP_514630.1;exception=RNA editing%2C trans-splicing;gbkey=CDS;gene=nad
NC_007886.1 RefSeq  CDS  315989 316246  .  -  0  ID=cds-YP_514630.1;Parent=gene-OrsaPp01;Dbxref=GenBank:YP_514630.1,GeneID:3950704;Name=YP_514630.1;exception=RNA editing%2C trans-splicing;gbkey=CDS;gene=nad
NC_007886.1 RefSeq  gene     17227  18069   .  -  .     ID=gene-OrsaPp02;Dbxref=GeneID:3950764;Name=cox3;gbkey=Gene;gene=cox3;gene_biotype=protein_coding;locus_tag=OrsaPp02
NC_007886.1 RefSeq  CDS  17227  18069   .  -  0  ID=cds-YP_514631.1;Parent=gene-OrsaPp02;Dbxref=GenBank:YP_514631.1,GeneID:3950764;Name=YP_514631.1;gbkey=CDS;gene=cox3;locus_tag=OrsaPp02;product=cytochrome
NC_007886.1 RefSeq  gene     18403  18996   .  -  .     ID=gene-OrsaPp03;Dbxref=GeneID:3950741;Name=orf25;gbkey=Gene;gene=orf25;gene_biotype=protein_coding;locus_tag=OrsaPp03
NC_007886.1 RefSeq  CDS  18403  18996   .  -  0  ID=cds-YP_514632.1;Parent=gene-OrsaPp03;Dbxref=GenBank:YP_514632.1,GeneID:3950741;Name=YP_514632.1;gbkey=CDS;gene=orf25;locus_tag=OrsaPp03;product=hypotheti
NC_007886.1 RefSeq  gene     20959  23258   .  +  .     ID=gene-OrsaPp04;Dbxref=GeneID:3950703;Name=ND5;exception=trans-splicing;gbkey=Gene;gene=ND5;gene_biotype=protein_coding;locus_tag=OrsaPp04;part=1
NC_007886.1 RefSeq  gene     233471 315625  .  -  .     ID=gene-OrsaPp04;Dbxref=GeneID:3950703;Name=ND5;exception=trans-splicing;gbkey=Gene;gene=ND5;gene_biotype=protein_coding;locus_tag=OrsaPp04;part=2
NC_007886.1 RefSeq  CDS  20959  21189   .  +  0  ID=cds-YP_514633.1;Parent=gene-OrsaPp04;Dbxref=GenBank:YP_514633.1,GeneID:3950703;Name=YP_514633.1;exception=trans-splicing;gbkey=CDS;gene=ND5;locus_tag=Orsa
NC_007886.1 RefSeq  CDS  22043  23258   .  +  0  ID=cds-YP_514633.1;Parent=gene-OrsaPp04;Dbxref=GenBank:YP_514633.1,GeneID:3950703;Name=YP_514633.1;exception=trans-splicing;gbkey=CDS;gene=ND5;locus_tag=Orsa
NC_007886.1 RefSeq  CDS  315605 315625  .  -  2  ID=cds-YP_514633.1;Parent=gene-OrsaPp04;Dbxref=GenBank:YP_514633.1,GeneID:3950703;Name=YP_514633.1;exception=trans-splicing;gbkey=CDS;gene=ND5;locus_tag=Orsa
NC_007886.1 RefSeq  CDS  233547 233941  .  -  2  ID=cds-YP_514633.1;Parent=gene-OrsaPp04;Dbxref=GenBank:YP_514633.1,GeneID:3950703;Name=YP_514633.1;exception=trans-splicing;gbkey=CDS;gene=ND5;locus_tag=Orsa
NC_007886.1 RefSeq  CDS  232471 232620  .  -  0  ID=cds-YP_514633.1;Parent=gene-OrsaPp04;Dbxref=GenBank:YP_514633.1,GeneID:3950703;Name=YP_514633.1;exception=trans-splicing;gbkey=CDS;gene=ND5;locus_tag=Orsa
NC_007886.1 RefSeq  gene     20959  118317  .  +  .     ID=gene-OrsaPp05;Dbxref=GeneID:3950702;Name=ND5;gbkey=Gene;gene=ND5;gene_biotype=protein_coding;locus_tag=OrsaPp05
NC_007886.1 RefSeq  CDS  20959  21189   .  +  0  ID=cds-YP_514634.1;Parent=gene-OrsaPp05;Dbxref=GenBank:YP_514634.1,GeneID:3950702;Name=YP_514634.1;exception=RNA editing;gbkey=CDS;gene=ND5;locus_tag=OrsaPp
NC_007886.1 RefSeq  CDS  117103 118317  .  +  0  ID=cds-YP_514634.1;Parent=gene-OrsaPp05;Dbxref=GenBank:YP_514634.1,GeneID:3950702;Name=YP_514634.1;exception=RNA editing;gbkey=CDS;gene=ND5;locus_tag=OrsaPp
NC_007886.1 RefSeq  gene     26691  26763   .  +  .     ID=gene-OrsaPt02;Dbxref=GeneID:3950726;Name=trnM(CAT);gbkey=Gene;gene=trnM(CAT);gene_biotype=tRNA;locus_tag=OrsaPt02
NC_007886.1 RefSeq  tRNA     26691  26763   .  +  .     ID=rna-OrsaPt02;Parent=gene-OrsaPt02;Dbxref=GeneID:3950726;Note=chloroplast-like;gbkey=tRNA;gene=trnM(CAT);locus_tag=OrsaPt02;product=tRNA-Met
NC_007886.1 RefSeq  exon     26691  26763   .  +  .     ID=exon-OrsaPt02-1;Parent=rna-OrsaPt02;Dbxref=GeneID:3950726;Note=chloroplast-like;gbkey=tRNA;gene=trnM(CAT);locus_tag=OrsaPt02;product=tRNA-Met
NC_007886.1 RefSeq  gene     32745  32818   .  -  .     ID=gene-OrsaPt03;Dbxref=GeneID:3950728;Name=trnH(GTG);gbkey=Gene;gene=trnH(GTG);gene_biotype=tRNA;locus_tag=OrsaPt03
NC_007886.1 RefSeq  tRNA     32745  32818   .  -  .     ID=rna-OrsaPt03;Parent=gene-OrsaPt03;Dbxref=GeneID:3950728;Note=chloroplast-like;gbkey=tRNA;gene=trnH(GTG);locus_tag=OrsaPt03;product=tRNA-His
NC_007886.1 RefSeq  exon     32745  32818   .  -  .     ID=exon-OrsaPt03-1;Parent=rna-OrsaPt03;Dbxref=GeneID:3950728;Note=chloroplast-like;gbkey=tRNA;gene=trnH(GTG);locus_tag=OrsaPt03;product=tRNA-His
NC_007886.1 RefSeq  gene     35764  35837   .  +  .     ID=gene-OrsaPt04;Dbxref=GeneID:3950730;Name=trnP(TGG);gbkey=Gene;gene=trnP(TGG);gene_biotype=tRNA;locus_tag=OrsaPt04
NC_007886.1 RefSeq  tRNA     35764  35837   .  +  .     ID=rna-OrsaPt04;Parent=gene-OrsaPt04;Dbxref=GeneID:3950730;Note=chloroplast-like;gbkey=tRNA;gene=trnP(TGG);locus_tag=OrsaPt04;product=tRNA-Pro
NC_007886.1 RefSeq  exon     35764  35837   .  +  .     ID=exon-OrsaPt04-1;Parent=rna-OrsaPt04;Dbxref=GeneID:3950730;Note=chloroplast-like;gbkey=tRNA;gene=trnP(TGG);locus_tag=OrsaPt04;product=tRNA-Pro
NC_007886.1 RefSeq  gene     35980  36053   .  +  .     ID=gene-OrsaPt05;Dbxref=GeneID:3950732;Name=trnW(CCA);gbkey=Gene;gene=trnW(CCA);gene_biotype=tRNA;locus_tag=OrsaPt05
NC_007886.1 RefSeq  tRNA     35980  36053   .  +  .     ID=rna-OrsaPt05;Parent=gene-OrsaPt05;Dbxref=GeneID:3950732;Note=chloroplast-like;gbkey=tRNA;gene=trnW(CCA);locus_tag=OrsaPt05;product=tRNA-Trp
NC_007886.1 RefSeq  exon     35980  36053   .  +  .     ID=exon-OrsaPt05-1;Parent=rna-OrsaPt05;Dbxref=GeneID:3950732;Note=chloroplast-like;gbkey=tRNA;gene=trnW(CCA);locus_tag=OrsaPt05;product=tRNA-Trp
```

Fonte: Autoria Própria (2025).

start - coordenada inicial, (5) *end* - coordenada final, (6) *score* - pontuação opcional, (7) *strand* - fita do DNA (’+’ ou ’-’), (8) *phase* - fase de leitura de sequencias codificantes (CDS - *coding sequences*), e (9) *attributes* - atributos adicionais no formato chave=valor. A indexação utiliza base-1 e as posições finais são inclusivas.

Com as anotações estruturadas no formato GFF3, torna-se possível identificar computacionalmente as regiões que não estão anotadas. Essas regiões, localizadas entre genes consecutivos (regiões intergênicas), são o alvo de extração da ferramenta.

1.3 “Região Intergênica”

No contexto deste trabalho, o termo “região intergênica” refere-se, em seu sentido estrito, aos intervalos existentes entre genes consecutivos em um genoma. Contudo, arquivos GFF3 abrangem uma ampla variedade de features além de genes codificadores de proteínas - como tRNA, rRNA, ncRNA, entre outras. Essa diversidade de anotações permite que o conceito de “região intergênica” seja reinterpretado de forma mais abrangente, conforme o escopo e os objetivos da análise.

Durante o desenvolvimento da versão *standalone* da ferramenta TIGRE, verificou-se a necessidade de incorporar tal flexibilidade conceitual. Para isso, a ferramenta permite o usuário especificar quais tipos de *features* presentes no GFF3 devem ser considerados na delimitação das regiões intergênicas. Essa funcionalidade viabiliza a extração de intervalos entre quaisquer ***features* de interesse**, permitindo que o usuário gere regiões intergênicas definidas, por exemplo, entre genes codificantes de proteína, entre tRNAs ou entre quaisquer combinações de elementos anotados, assegurando que cada usuário possa **ajustar a definição de “região intergênica”** às particularidades de sua investigação.

1.4 Anotação de regiões intergênicas

Quando uma região intergênica é extraída, ela é anotada no formato GFF3 incluindo informações sobre as *features* que a delimitam. A TIGRE identifica a *feature* imediatamente anterior (chamada de *upstream*) e a *feature* imediatamente posterior (chamada de *downstream*) à região intergênica de interesse.

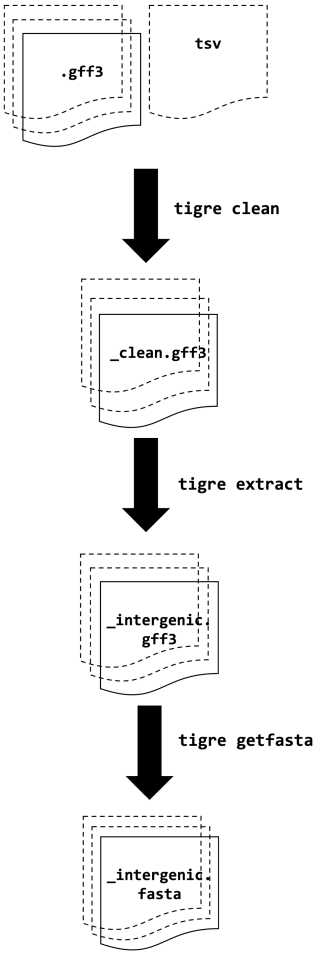
Essas informações são armazenadas nos atributos da região intergênica extraída, permitindo rastrear o contexto genômico de cada intervalo e facilitando análises posteriores que necessitem identificar quais *features* flanqueiam uma determinada região intergênica.

Com a compreensão dos conceitos fundamentais de anotações genômicas, do formato GFF3 e da flexibilidade na definição de regiões intergênicas, é possível detalhar como a TIGRE foi implementada para atender a essas necessidades. O próximo capítulo apresenta a arquitetura da ferramenta, suas funcionalidades e o fluxo de processamento dos dados.

2 DESENVOLVIMENTO

A TIGRE foi implementada como CLI em *Python*, projetada para ser integrada em *pipelines* de bioinformática e executada em ambientes *Linux* ou *macOS*. A escolha por uma interface de linha de comando permite maior automação, facilita o processamento em lote de múltiplos genomas e possibilita a integração com outras ferramentas e *scripts* de análise.

Figura 2 – Diagrama do processo em modo *multiple*.



Fonte: Adaptado da documentação da TIGRE.

Conforme ilustrado na Figura 2, a TIGRE estrutura seu processamento em um fluxo sequencial composto por três etapas principais, que podem ser executadas tanto sobre um único genoma (modo *single*) quanto sobre múltiplos genomas (modo *multiple*). Independentemente do modo utilizado, o fluxo permanece o mesmo: inicialmente, a etapa `clean` realiza a preparação e padronização dos arquivos GFF3, filtrando as *features* de interesse e resolvendo sobreposições. Em seguida, a etapa `extract` aplica a lógica de delimitação das regiões intergênicas sobre os arquivos já processados. Por fim, a etapa `getfasta` recupera as sequências de nucleotídeos correspondentes a cada intervalo identificado, produzindo os arquivos FASTA necessários para análises subsequentes.

Cada comando opera de forma independente, permitindo que o usuário execute apenas as etapas necessárias para sua análise específica ou utilize os resultados intermediários para outras finalidades.

Este capítulo apresenta os aspectos técnicos da implementação da TIGRE. Inicia-se com a descrição da arquitetura de processamento e tecnologias utilizadas, incluindo os dois modos de operação (*single* e *multiple*). Em seguida, apresenta-se a estrutura do código fonte e organização do projeto. O sistema de *logging* é então detalhado, seguido pela especificação completa de cada um dos três comandos principais, incluindo suas *flags*, modos de operação e lógica de processamento. O capitulo é finalizado com as limitações da ferramenta.

2.1 Arquitetura e Tecnologias

A TIGRE foi desenvolvida em *Python* 3.12, priorizando o mínimo de dependências externas para facilitar instalação e manutenção em diferentes ambientes computacionais. A Tabela 1 apresenta as dependências do projeto e suas versões mínimas requeridas, enquanto a Tabela 2 apresenta as fontes de cada recurso utilizado.

Tabela 1 – Dependências da ferramenta TIGRE.

Dependência	Versão	Requerimento
pandas	>=1.5.3, <3.0.0	Obrigatória
biopython	>=1.80	Opcional, necessária apenas para o comando `getfasta`
gdt	>=1.0.1	Opcional, necessária somente ao usar a flag `--gdict` no comando `clean`

Fonte: Autoria Própria (2025).

Tabela 2 – Recursos usados no desenvolvimento ferramenta.

Recurso	Fonte	Site Oficial
Python 3.12	*Python Software Foundation*	https://python.org
pandas	*The Pandas Development Team*	https://pandas.pydata.org
biopython	Cock *et al.* (2009)	https://biopython.org
gdt	Dupin *et al.* (2025)	https://github.com/brenodupin/gdt

Fonte: Autoria Própria (2025).

A TIGRE implementa cada comando (`clean`, `extract`, `getfasta`) como uma **função isolada** que processa um único genoma por vez. Cada genoma é identificado por seu número de acesso do NCBI (*accession number*, AN). Por exemplo, na Figura 1, o AN é NC_-007886.1. Essas funções recebem os caminhos dos arquivos de entrada e saída como parâmetros e executam o processamento de forma independente. Essa arquitetura possibilita dois modos de operação:

2.1.1 Modo *Single*

Executa a função uma única vez com caminhos de arquivos especificados diretamente pelo usuário via *flags* (`--gff-in`, `--gff-out`, etc.). Este modo é útil para processar genomas individuais ou quando se deseja controle explícito sobre os caminhos dos arquivos.

2.1.2 Modo *Multiple*

O modo *multiple* foi projetado para análises em larga escala, permitindo o processamento paralelo de dezenas de genomas simultaneamente. O usuário fornece um arquivo TSV contendo uma coluna com ANs (por padrão, a coluna é nomeada "AN", mas pode ser customizada via *flag* `--an-column`). A estrutura esperada do projeto segue a convenção:

```
projeto/
    genomas.tsv
    NC_007886.1/
        NC_007886.1.gff3
        NC_007886.1.fasta (opcional)
    NC_123456.1/
        NC_123456.1.gff3
        NC_123456.1.fasta (opcional)
```

Cada AN listado no TSV deve ter um diretório correspondente no mesmo nível do arquivo TSV. A TIGRE constrói automaticamente os caminhos dos arquivos seguindo o padrão: `<AN>/<AN><sufixo><extensão>`

Os sufixos e extensões são configuráveis através de *flags* específicas de cada comando e modo. Por exemplo, no comando *clean*, o sufixo padrão de entrada é vazio (`--gff-in-suffix ""`) e o de saída é "_clean"(`--gff-out-suffix "_clean"`). Assim, para o AN NC_007886.1, a TIGRE leria `NC_007886.1/NC_007886.1.gff3` e geraria `NC_007886.1/NC_007886.1_clean.gff3`.

Essa convenção facilita o encadeamento dos comandos em uma *pipeline*, onde a saída de um comando se torna a entrada do próximo através do ajuste dos sufixos. O processamento é distribuído automaticamente entre múltiplos processos (quantidade configurável via `--workers`), onde cada *worker* processa um AN de forma independente.

2.2 Código Fonte

A TIGRE é uma ferramenta de código aberto (*open source*) distribuída sob a licença MIT, permitindo uso, modificação e redistribuição livres. O código fonte está disponível publicamente no *GitHub* em https://github.com/brenodupin/tigre, onde também são mantidos a documentação completa, *issues*, e exemplos de uso. O desenvolvimento utilizou *Git* como sistema de controle

de versão, com todo o histórico de *commits* preservado no repositório, permitindo rastreabilidade completa do processo de desenvolvimento.

2.2.1 Instalação e Distribuição

A ferramenta está disponível no PyPI (*Python Package Index*), permitindo instalação simplificada via `pip`. A versão atual (1.0.3) pode ser instalada de diferentes formas, dependendo das funcionalidades desejadas:

- `>pip install tigre`: Instalação básica com dependências mínimas.
- `>pip install "tigre[gdt]"`: Inclui *gdt* para a *flag* `--gdt` no comando *clean*.
- `>pip install "tigre[bio]"`: Inclui *biopython* para o comando *getfasta*.
- `>pip install "tigre[all]"`: Instalação completa com todas as dependências opcionais.

Essa flexibilidade permite que usuários instalem apenas o necessário para seus casos de uso específicos, minimizando a instalação das dependências.

2.2.2 Dados de Exemplo e Reprodutibilidade

O repositório *GitHub* inclui um *dataset* de exemplo contendo 1.207 genomas mitocondriais de plantas, todos provenientes de dados públicos do NCBI. Este *dataset* é utilizado para demonstração da *pipeline* completo e validação da ferramenta, e será empregado no Capítulo 3 deste trabalho para ilustrar a aplicação prática da TIGRE.

Para garantir a preservação a longo prazo, a TIGRE possui um DOI (*Digital Object Identifier*) através da plataforma Zenodo, disponível em https://zenodo.org/records/17475986, onde versões específicas da ferramenta são arquivadas permanentemente, facilitando a reprodutibilidade de análises científicas.

2.2.3 Qualidade de Código

O código fonte da TIGRE segue padrões estabelecidos pela comunidade *Python* para garantir qualidade e manutenibilidade. O projeto adota o estilo de formatação *Black*, que proporciona consistência visual automática ao código, e utiliza *Ruff* para *linting*, identificando potenciais problemas e más práticas de programação.

Para garantir robustez e detectar erros em tempo de desenvolvimento, a TIGRE utiliza verificação estrita de tipos através do *mypy* com a *flag* `--strict`, exigindo anotações de tipo completas em todo o código. Essa abordagem aumenta a confiabilidade da ferramenta ao identificar potenciais erros antes da execução.

A qualidade do código é assegurada através de dois *workflows* de Integração Contínua (CI) implementados via *GitHub Actions*. O primeiro *workflow* executa verificações a cada *push* ou *pull request* que modifica o código fonte: *linting* com *Ruff*, verificação de formatação com *Black*, e checagem de tipos com *mypy*. Este *workflow* sinaliza problemas mas não impede a integração ao repositório, permitindo flexibilidade durante o desenvolvimento.

O segundo *workflow*, acionado ao criar uma nova *tag* de versão (formato `v*`), garante qualidade na publicação de novas versões. Antes de construir e publicar o pacote no PyPI, todas as três verificações são executadas obrigatoriamente. Apenas código que passa em todas as verificações é publicado, assegurando que usuários recebam versões estáveis e de alta qualidade. Este *workflow* também gera automaticamente um *changelog* e cria uma *release* no *GitHub* com as mudanças desde a versão anterior.

2.3 Sistema de *logging*

A TIGRE implementa um sistema robusto de *logging* que permite rastreamento detalhado da execução e diagnóstico de problemas. Por padrão, os *logs* são salvos em arquivos nomeados no formato `tigre_<comando>-<timestamp>.log` no diretório atual de trabalho.

O sistema segue o padrão da biblioteca *logging* do *Python*, utilizando diferentes níveis hierárquicos de informação: *DEBUG*, *INFO*, *WARNING* e *ERROR*. A TIGRE estende esse padrão com a adição do nível *TRACE*, mais detalhado que *DEBUG*, para permitir rastreamento extremamente granular do fluxo de execução.

O controle de verbosidade é realizado através da *flag* `-v` (ou `--verbose`), que pode ser especificada múltiplas vezes para aumentar o nível de detalhamento:

- Sem *flag*: Nível *INFO* no console, nível *DEBUG* no arquivo.
- `-v`: Nível *INFO* no console, nível *TRACE* no arquivo.
- `-vv`: Nível *DEBUG* no console, nível *TRACE* no arquivo.
- `-vvv`: Nível *TRACE* em ambos (console e arquivo).

Flags adicionais permitem customização do comportamento de *logging*:

- `--log <caminho>`: Define um caminho personalizado para o arquivo de *log*.
- `--no-log-file`: Desabilita completamente o registro em arquivo.
- `--quiet`: Suprime toda saída no console.

Essa flexibilidade é especialmente útil no modo *multiple*, onde o processamento paralelo de múltiplos genomas pode gerar grande volume de informações. O sistema de *logging* facilita a identificação de erros específicos em ANs individuais e permite auditoria completa do processamento realizado.

2.4 Comandos

A TIGRE é organizada em três comandos principais - `clean`, `extract` e `getfasta`, executados sempre como `tigre clean [options...]`, `tigre extract [options...]` e `tigre getfasta [options...]`.

Figura 3 – Saída gerada por `tigre --help`.

```
❯ tigre --help
usage: tigre [-h] [-v] [--log PATH] [--no-log-file] [--quiet] [--version] {clean,extract,getfasta} ...

 T   I   G   E   R   \    /\
 |   |   |   | x |    )  ( ')
 T   I   G   R   E   (  /  )
 o   n   e   e   x    \(__)|
 o   t   n   g   t     jgs
 l   e   i    i   r
     r   c     o   a c t i o n
 f              n
 o
 r

positional arguments:
  {clean,extract,getfasta}
    clean               Clean, standardize, and prepare GFF3 file(s) for processing.
    extract             Extract intergenic regions from processed GFF3 file(s).
    getfasta            Generate FASTA sequences from intergenic regions.

options:
  -h, --help            show this help message and exit
  --version             Show the version of the tigre package.

log options:
  -v, --verbose         Increase verbosity level. Use multiple times for more verbose output: -v (INFO
                        console, TRACE file), -vv (DEBUG console, TRACE file), -vvv (TRACE console,
                        TRACE file). Default: INFO console + DEBUG file.
  --log PATH            Path to the log file. If not provided, a default log file will be created.
  --no-log-file         Disable file logging.
  --quiet               Suppress console output.
```

Fonte: Autoria Própia (2025).

A Figura 3 apresenta o *help message* da TIGRE, no qual são listados os comandos disponíveis, suas descrições e as principais opções de execução. O usuário pode usar a *flag* `--help` em todas as partes do comando, por exemplo, `tigre --help`, `tigre clean --help`, `tigre clean multiple --help` ou `tigre clean single --help`. Servindo como uma referência rápida as *flags* disponíveis.

Nos tópicos seguintes, cada comando é discutido em detalhe, incluindo sua função, parâmetros, opções relevantes e lógica de processamento.

2.4.1 *clean*

O comando *clean* é responsável pela preparação e padronização de arquivos GFF3, constituindo a primeira etapa do pipeline de processamento. O comando processa os arquivos retendo apenas as *features* que correspondem aos critérios definidos em `--query-string`,

resolvendo anotações sobrepostas e, opcionalmente, normalizando nomes das *features* utilizando a ferramenta GDT.

Flags específicas:

- `--query-string <string>`: Determina quais *features* são selecionadas como região de interesse. (padrão: `type in ('gene', 'tRNA', 'rRNA', 'region')`).
- `--gdict <caminho>`: Arquivo .gdict da ferramenta GDT para padronização de nomenclatura de genes. Opcional.
- `--keep-orfs`: Mantém sequências ORFs no arquivo de saída. Por padrão, a TIGRE remove todas as ORFs.
- `--extended-filtering`: Ativa filtragem estendida para remover ORFs e suas features relacionadas de forma mais agressiva. Pode remover mais *features* do que o pretendido em alguns casos.
- `--overwrite`: Sobrescreve arquivos de saída existentes. Por padrão, a TIGRE não executa se algum arquivo de saída já existe, prevenindo sobrescrita acidental.

Flags do modo *Single*:

- `--gff-in <caminho>`: Caminho do arquivo GFF3 de entrada.
- `--gff-out <caminho>`: Caminho do arquivo GFF3 de saída.

Flags do modo *Multiple*:

- `--tsv <caminho>`: Arquivo TSV com ANs.
- `--an-column <string>`: Nome da coluna com números de acesso no TSV (padrão: "AN").
- `--workers <int>`: Número de *workers* para processamento paralelo (padrão: 0, utiliza todos os núcleos disponíveis).
- `--gff-in-suffix <string>`: Sufixo dos arquivos GFF3 de entrada (padrão *string* vazia).
- `--gff-in-ext <string>`: Extensão dos arquivos GFF3 de entrada (padrão: ".gff3").
- `--gff-out-suffix <string>`: Sufixo dos arquivos GFF3 de saída (padrão: "_clean").
- `--gff-out-ext <string>`: Extensão dos arquivos GFF3 de saída (padrão: ".gff3").

Otimização de memória com GDT: Quando o comando *clean* é executado com um arquivo GDT no modo *multiple*, a TIGRE automaticamente implementa uma arquitetura servidor-cliente para otimizar o uso de memória. Um processo servidor dedicado carrega e consulta o dicionário de genes (.gdict), enquanto os outros processos se comunicam com ele via filas e *pipes* de comunicação entre processos para realizar buscas de nomenclatura. Esse modelo evita que cada processo carregue sua própria cópia do arquivo .gdict, que pode ser grande, reduzindo significativamente o consumo total de memória.

A TIGRE *clean* lida com anotações complexass incluindo *features* sobrepostas e limites de genomas circulares. Padronizar e/ou resolver essas complexidades é importante, pois anotações sobrepostas e genomas circulares podem dificultar computacionalmente a identificação e extração de regiões intergênicas.

2.4.1.1 *Features* que atravessam limites de genomas circulares

Em genomas circulares, algumas *features* podem atravessar o "limite" do genoma, iniciando antes da coordenada final e estendendo-se além da coordenada inicial.

O comando *clean* resolve esse problema dividindo cada *feature* que atravessa o limite em duas anotações separadas, adicionando o sufixo `_fragment` ao tipo da *feature* no GFF3. A divisão é realizada da seguinte forma:

- **Primeiro fragmento:** Abrange desde a coordenada inicial da *feature* original até o final do genoma.
- **Segundo fragmento:** Abrange desde o início do genoma (coordenada 1) até a posição final original da *feature*, menos o tamanho do genoma.

Ambos os fragmentos mantêm todos os atributos da *feature* original. Essa estratégia de fragmentação simplifica as etapas subsequentes ao permitir que o genoma circular seja inicialmente tratado como linear, lidando com a circularidade apenas ao processar regiões próximas ao limite do genoma. Na subseção 2.4.2.1 é apresentada a Figura 5, onde no cenário (d) acontece uma fragmentação do gene MIT-ATP6.

2.4.1.2 Resolução de *features* sobrepostas

Quando o comando clean detecta que múltiplas *features* se sobrepõem, ele cria um novo tipo de *feature* chamado `overlapping_feature_set`. Essa *feature* especial marca regiões onde nenhuma sequência intergênica válida pode existir devido à sobreposição de *features*. O `overlapping_feature_set` é gerado pela fusão de todas as *features* sobrepostas em uma única anotação consolidada. Sua fita (*strand*) é sempre definida como '+', uma vez que representa uma metarregião combinada, e não uma *feature* em uma fita específica.

Para preservar o contexto para análises subsequentes, o `overlapping_feature_set` armazena informações identificadoras apenas das *features* nos limites da re-

gião sobreposta. Essas *features* de limite são utilizadas para determinar os delimitadores *upstream* e *downstream* das regiões intergênicas adjacentes. Os atributos do `overlapping_feature_set` incluem:

- `name_left`: Obtido do valor `ID` na coluna de atributos (ou nomeclatura padronizada pelo GDT) da *feature* que inicia no limite esquerdo da região sobreposta.
- `source_left`: Valor customizado construído usando as informações originais da *feature* no formato: `seqid|type|start|end|strand|ID`.
- `name_right`: Obtido do valor `ID` na coluna de atributos (ou nomeclatura gene padronizada pelo GDT) da *feature* que termina no limite direito da região sobreposta.
- `source_right`: Valor customizado construído usando as informações originais da *feature* no formato: `seqid|type|start|end|strand|ID`.

Quando existem múltiplas *features* que iniciam ou terminam nos limites da região sobreposta, o comando *clean* utiliza as seguintes regras para selecionar as *features* de limite:

- **Esquerda:** Escolhe a *feature* com a menor coordenada de início. Em caso de empate, seleciona a *feature* com a maior coordenada de término.
- **Direita:** Escolhe a *feature* com a maior coordenada de término. Em caso de empate, seleciona a *feature* com a menor coordenada de início.

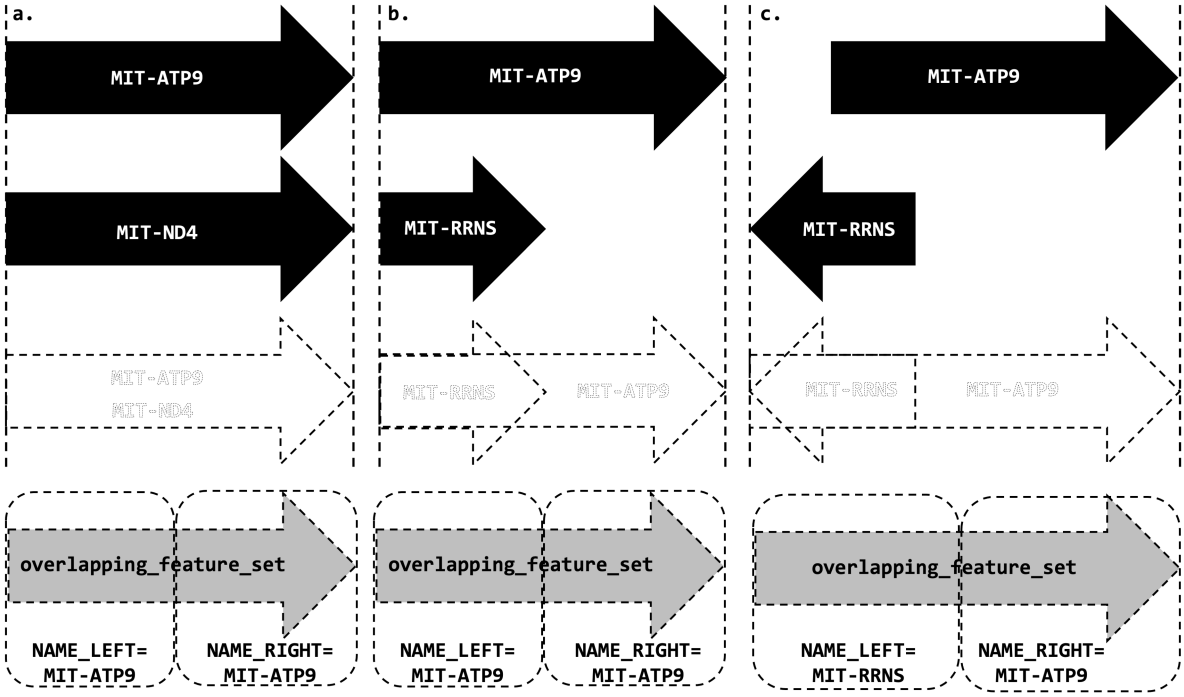
Se duas *features* possuem coordenadas de início e término idênticas, a que aparece primeiro no arquivo GFF3 é escolhida.

A Figura 4 ilustra três cenários possíveis de sobreposição de *features*. Em (a), as *features* MIT-ATP9 e MIT-ND4 possuem exatamente as mesmas coordenadas de início e término, sobrepondo-se completamente. Neste caso, como a TIGRE não corrige anotações, o algoritmo seleciona arbitrariamente a primeira *feature* listada no arquivo GFF3 para as informações de limite, e o `overlapping_feature_set` resultante coincide com a *feature* escolhida.

Em (b), há sobreposição entre uma *feature* maior (MIT-ATP9) e uma menor (MIT-RRNS) que compartilham a mesma coordenada de início. A resolução do limite esquerdo segue a regra de desempate: MIT-ATP9 é escolhida por ter a maior coordenada de término. Para o limite direito, MIT-ATP9 também é selecionada por ser a única *feature* que termina no limite direito da região sobreposta.

Em (c), ocorre uma sobreposição regular entre MIT-RRNS e MIT-ATP9, onde a região fundida torna-se um `overlapping_feature_set` com *features* de limite determinadas pelas regras de seleção (menor coordenada de início para o limite esquerdo, maior coordenada de término para o limite direito). Em todos os casos, o `overlapping_feature_set` (representado pela área cinza) consolida as regiões sobrepostas, preservando as informações das *features* nos limites para determinar as regiões intergênicas adjacentes.

Figura 4 – Cenários de resolução de *features* sobrepostas.



Fonte: Adaptado da documentação da TIGRE.

Note que a fita (*strand*) do `overlapping_feature_set` é sempre definida como '+', independentemente das fitas das *features* originais. Entretanto, essa informação não é descartada: no caso (c), por exemplo, a fita original de MIT-RRNS é preservada no atributo `source_left` do `overlapping_feature_set`, permitindo rastreabilidade completa das *features* originais que compõem a região sobreposta.

A implementação da resolução de sobreposições é descrita no pseudocódigo apresentado a seguir. A função recebe três parâmetros: `log`, que gerencia o *logging* (seção 2.3); `df`, contendo as *features* já filtradas e ordenadas pela coordenadas de início; e `linha_região`, que corresponde à linha do arquivo GFF3 que representa a região total do genoma:

```
FUNÇÃO overlap_solver(log, df, linha_região)
    próximo_índice ← 0
    resultado ← [] // lista vazia

    PARA CADA linha EM df FAÇA
        SE índice_atual É DIFERENTE DE próximo_índice ENTÃO
            CONTINUAR  // pula para próxima linhas
        FIM SE

        fim_região ← linha.fim
        próximo_índice ← próximo_índice + 1
        sobreposições ← [] // lista vazia
```

```
        // Detecta sobreposições com linhas seguintes
        ENQUANTO (próximo_índice MENOR QUE tamanho(df)) E
                 (fim_região MAIOR OU IGUAL A início_da_próxima_linha)

            ADICIONAR próxima_linha EM sobreposições

            SE fim_região MENOR QUE fim_da_próxima_linha ENTÃO
                fim_região ← fim_da_próxima_linha
            FIM SE

            próximo_índice ← próximo_índice + 1
        FIM ENQUANTO

        // Processa sobreposições encontradas
        SE sobreposições NÃO ESTÁ VAZIA ENTÃO
            ADICIONAR linha NO INÍCIO DE sobreposições

            // Transforma a linha atua na metaregião
            linha.fim ← fim_região
            linha.tipo ← "overlapping_feature_set"
            linha.atributos ← escolher_bordas(sobreposições)
        FIM SE

        ADICIONAR linha EM resultado
    FIM PARA

    df_region ← concatenar(linha_região, resultado)

    RETORNAR df_region
FIM FUNÇÃO
```

Com os arquivos GFF3 padronizados e preparados pelo comando *clean*, torna-se possível identificar e extrair as regiões intergênicas propriamente ditas. O comando *extract* executa essa tarefa, identificando os intervalos entre *features* consecutivas e gerando novas anotações para essas regiões.

2.4.2 *extract*

O comando *extract* identifica e extrai as regiões intergênicas dos arquivos GFF3 processados pelo *clean*, criando novas anotações para os intervalos entre *features*. Embora este

comando execute a extração propriamente dita, o objetivo principal da TIGRE, todo o trabalho computacional intensivo foi realizado previamente pelo *clean*. A ordenação, resolução de sobreposições e fragmentação de *features* que atravessam limites circulares (subseção 2.4.1.1) simplificam significativamente a lógica do *extract*, que essencialmente identifica os "espaços negativos"entre as *features* já preparadas.

O comando *extract* espera arquivos GFF3 contendo *features* completamente independentes, com sobreposições e *features* que atravessam limites já resolvidas. Todas as *features* devem estar ordenadas por coordenada de início e seus atributos padronizados no formato `name=value;source=value;` (ou `name_left=value;source_left=value;name_right=value;source_right=value;` para `overlapping_feature_set`).

É importante notar que, embora o termo “região intergênica” seja utilizado por padrão, o usuário pode customizar o tipo de *feature* usado para representar essas regiões através da *flag* `--feature-type`.

Flags específicas:

- `--feature-type <string>`: Nome do tipo de *feature* para as regiões intergênicas (padrão: "intergenic_region").
- `--add-region`: Adiciona linha de região ao arquivo GFF3 de saída.
- `--overwrite`: Sobrescreve arquivos de saída existentes.

Flags do modo Single:

- `--gff-in <caminho>`: Caminho do arquivo GFF3 de entrada.
- `--gff-out <caminho>`: Caminho do arquivo GFF3 de saída.

Flags do modo *Multiple*:

- `--tsv <caminho>`: Arquivo TSV com números de acesso.
- `--an-column <string>`: Nome da coluna com números de acesso no TSV (padrão: "AN").
- `--workers <int>`: Número de *workers* para processamento paralelo (padrão: 0, utiliza todos os núcleos disponíveis).
- `--gff-in-suffix <string>`: Sufixo dos arquivos GFF3 de entrada (padrão: "_clean").
- `--gff-in-ext <string>`: Extensão dos arquivos GFF3 de entrada (padrão: ".gff3").
- `--gff-out-suffix <string>`: Sufixo dos arquivos GFF3 de saída (padrão: "_intergenic").

- `--gff-out-ext <string>`: Extensão dos arquivos GFF3 de saída (padrão: ".gff3").

A função que implementa o comando `extract` identifica intervalos entre *features* consecutivas e cria anotações de regiões intergênicas para esses intervalos. Cada região intergênica é definida como a sequência entre o final de uma *feature* e o início da próxima. Portanto, sua coordenada de início é uma posição após o final da *feature upstream* (esquerda), e sua coordenada de término é uma posição antes do início da *feature downstream* (direita).

Cada região intergênica captura informações sobre suas *features* flanqueadoras em seus atributos:

- `name_up` e `source_up`: Informações identificadoras da *feature* flanqueadora *upstream*
- `name_dw` e `source_dw`: Informações identificadoras da *feature* flanqueadora *downstream*

2.4.2.1 Tratamento de limites do genoma

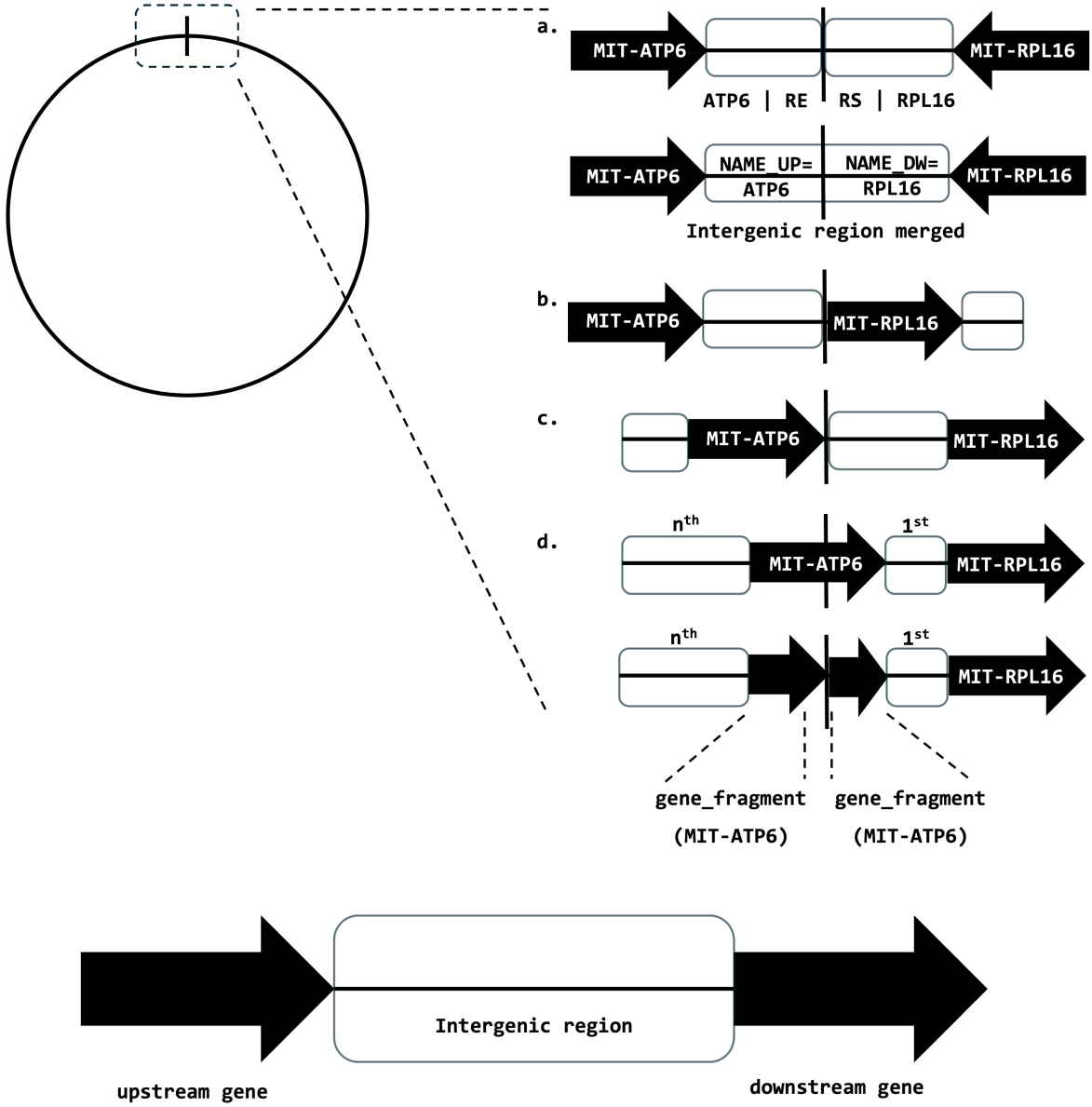
O comando *extract* implementa lógica especial para garantir que regiões intergênicas nos limites inicial e final do genoma sejam corretamente identificadas, especialmente para genomas circulares.

Regiões no limite inicial: Se a primeira *feature* do genoma não inicia na coordenada 1, uma região intergênica é criada da posição 1 até o início da primeira *feature*. Para essa região de limite, `name_up` e `source_up` são derivados da última *feature* se o genoma for circular, ou definidos como `region_start` se linear. Os atributos `name_dw` e `source_dw` são derivados da primeira *feature*.

Regiões no limite final: Se a última *feature* não termina na coordenada final do genoma, uma região intergênica é criada do final da última *feature* até o final do genoma. Para essa região, `name_up` e `source_up` são derivados da última *feature*, enquanto `name_dw` e `source_dw` são derivados da primeira *feature* se o genoma for circular, ou definidos como `region_end` se linear.

Fusão de limites em genomas circulares: Para genomas circulares (identificados pela presença de `is_circular=true` nos atributos da *feature region*), o *extract* realiza uma verificação adicional. Se tanto uma região de limite inicial quanto uma de limite final seriam criadas (ou seja, a primeira *feature* inicia após a coordenada 1 **e** a última *feature* termina antes do final do genoma), uma única região intergênica que atravessa o limite do genoma é criada. Essa região fundida abrange do final da última *feature*, atravessa o limite do genoma e continua até o início da primeira *feature*. Seu tipo de feature recebe o sufixo `_merged` para indicar que cruza o limite do genoma, e utiliza `name_up`/`source_up` da última *feature* e `name_dw`/`source_dw` da primeira *feature*. Essa lógica de fusão reconhece que, em genomas circulares, o espaço no final e no início do genoma forma, na realidade, uma única região intergênica contínua.

Figura 5 – Cenários de extração de regiões intergênicas em um genoma circular.



Fonte: Adaptado da documentação da TIGRE.

A Figura 5 ilustra diferentes cenários de extração de regiões intergênicas em um genoma circular mitocondrial. A parte inferior da figura apresenta a representação esquemática geral de uma região intergênica, mostrando como ela é delimitada por uma *feature upstream* e outra *downstream*.

No cenário (a), é demonstrado o caso de fusão de limites em genoma circular. A feature MIT-ATP6 termina próximo ao final do genoma, enquanto MIT-RPL16 inicia próximo ao início. O *extract* identifica que há espaço tanto no final (RE, *region end*) quanto no início (RS, *region start*) do genoma. Como o genoma é circular, essas duas regiões são fundidas em uma única `intergenic_region_merged` que atravessa o limite do genoma, com `name_up=ATP6` (da última *feature*) e `name_dw=RPL16` (da primeira *feature*).

O cenário (b) mostra uma situação onde MIT-ATP6 termina próximo ao final do genoma, mas MIT-RPL16 inicia exatamente na coordenada 1. Neste caso, apenas uma região intergênica no limite final é criada, com MIT-ATP6 como *feature upstream* e MIT-RPL16 como *downstream*.

O cenário (c) apresenta o caso inverso: MIT-ATP6 termina exatamente na última coordenada do genoma, enquanto MIT-RPL16 inicia após a coordenada 1. Aqui, apenas uma região intergênica no limite inicial é criada.

No cenário (d), é ilustrado o tratamento de *features* que atravessam o limite do genoma. A *feature* MIT-ATP6 foi fragmentada pelo comando *clean* (como detalhado na subseção 2.4.1.1) em dois `gene_fragment`, um no final do genoma e outro no início. O *extract* utiliza essas informações dos fragmentos para criar corretamente as regiões intergênicas adjacentes. A região intergênica n^{th} (última) tem MIT-ATP6 (fragmento final) como *feature downstream*, enquanto a região intergênica 1^{st} (primeira) tem MIT-ATP6 (fragmento inicial) como *feature upstream*. Não há região intergênica entre os dois fragmentos, pois eles são consecutivos e representam a mesma *feature* original. Dessa forma, as informações da *feature* MIT-ATP6 que atravessa o limite são corretamente propagadas como delimitadores das regiões intergênicas adjacentes através de seus fragmentos.

Até este ponto, a TIGRE identificou e anotou as posições das regiões intergênicas no genoma. Para análises subsequentes que necessitem das sequências de nucleotídeos dessas regiões, o comando *getfasta* extrai essas sequências dos arquivos FASTA de referência.

2.4.3 *getfasta*

O comando *getfasta* recupera as sequências de nucleotídeos das regiões intergênicas extraídas pelo comando *extract*. As sequências são salvas em arquivos multi-FASTA (um por AN). Este comando requer a dependência opcional biopython.

Flags específicas:

- `--bedtools-compatible`: Utiliza indexação base-0 nos cabeçalhos FASTA para compatibilidade com *bedtools*.
- `--overwrite`: Sobrescreve arquivos de saída existentes.

Flags do modo *Single*:

- `--gff-in <caminho>`: Caminho do arquivo GFF3 de entrada.
- `--fasta-in <caminho>`: Caminho do arquivo FASTA de entrada (genoma completo).
- `--fasta-out <caminho>`: Caminho do arquivo FASTA de saída.

Flags do modo *Multiple*:

- `--tsv <caminho>`: Arquivo TSV com números de acesso.

- `--an-column <string>`: Nome da coluna com números de acesso no TSV (padrão: "AN").
- `--workers <int>`: Número de *workers* para processamento paralelo (padrão: 0, utiliza todos os núcleos disponíveis).
- `--gff-in-suffix <string>`: Sufixo dos arquivos GFF3 de entrada (padrão: "_intergenic").
- `--gff-in-ext <string>`: Extensão dos arquivos GFF3 de entrada (padrão: ".gff3").
- `--fasta-in-suffix <string>`: Sufixo dos arquivos FASTA de entrada (padrão: "").
- `--fasta-in-ext <string>`: Extensão dos arquivos FASTA de entrada (padrão: ".fasta").
- `--fasta-out-suffix <string>`: Sufixo dos arquivos FASTA de saída (padrão: "_intergenic").
- `--fasta-out-ext <string>`: Extensão dos arquivos FASTA de saída (padrão: ".fasta").

2.4.3.1 Formato do cabeçalho FASTA

Os cabeçalhos dos arquivos FASTA são salvos com o seguinte formato: `>type::seqid:start-end`

Onde:

- `type`: Tipo da *feature* (ex: `intergenic_region`, `intergenic_region_merged`)
- `seqid`: Coluna *seqid* do arquivo GFF3 (geralmente o AN).
- `start`: Posição inicial da região intergênica.
- `end`: Posição final da região intergênica.

Por padrão, a TIGRE utiliza indexação base-1 nos cabeçalhos das sequências, seguindo as convenções do GFF3. Quando a *flag* `--bedtools-compatible` é ativada, os cabeçalhos utilizam indexação base-0 para integração da TIGRE em *pipelines* que utilizam o *bedtools*, enquanto as sequências em si permanecem inalteradas.

2.5 Limitações

É importante ressaltar que, uma vez que a TIGRE utiliza as anotações do arquivo GFF3 como base para identificar e extrair as regiões intergênicas, a qualidade e a precisão dos resultados obtidos estão vinculadas à **qualidade das anotações genômicas fornecidas**. Anotações incompletas, imprecisas ou desatualizadas podem resultar na identificação incorreta de regiões intergênicas. A ferramenta **não realiza correção ou validação das anotações presentes no arquivo de entrada**, atuando exclusivamente como um extrator baseado nas informações fornecidas.

Para mitigar potenciais problemas decorrentes de anotações de baixa qualidade, recomenda-se a utilização de arquivos GFF3 provenientes de fontes confiáveis e bem estabelecidas, como o banco de dados RefSeq do NCBI, que oferece anotações genômicas curadas e de alta qualidade.

Com a implementação completa da TIGRE apresentada, incluindo sua arquitetura, código fonte e funcionalidades de cada comando, o próximo capítulo demonstra a aplicação prática da ferramenta. Utilizando o *dataset* de genomas mitocondriais de plantas descrito na subseção 2.2.2, valida-se o funcionamento e desempenho da TIGRE em um cenário real de análise em larga escala.

3 APLICAÇÃO PRÁTICA

Este capítulo demonstra a aplicação da TIGRE em um cenário real de análise genômica em larga escala, utilizando o *dataset* de genomas mitocondriais de plantas descrito na subseção 2.2.2. O objetivo é ilustrar o funcionamento completo da *pipeline*, desde a preparação do ambiente até a obtenção das sequências intergênicas, validando a eficácia da ferramenta no processamento de grandes volumes de dados.

3.1 Preparação do Ambiente

O primeiro passo consiste em obter o *dataset* de exemplo e preparar o ambiente para utilização da TIGRE. O conjunto de dados utilizado nesta documentação encontram-se no repositório oficial da ferramenta, disponível em https://github.com/brenodupin/tigre. A ferramenta foi desenvolvida em ambiente *Linux* (*Ubuntu*) e testada em *macOS*.

Para baixá-lo, acesse o repositório, navegue até *examples* e faça o download direto do arquivo `plants_mit.tar.zst`. Depois, proceda com a descompactação e extração:

```
>zstd -d -c plants_mit.tar.zst | tar -xf -
>cd plants_mit
```

Após a extração, o diretório `plants_mit` contém a seguinte estrutura:

- Arquivo TSV (`plants_mit.tsv`) com a lista dos 1.207 números de acesso.
- Arquivo .gdict (`plants_mit.gdict`) para padronização de nomenclatura, através da ferramenta GDT.
- Diretórios individuais para cada genoma, contendo arquivos GFF3 e FASTA.

A instalação da TIGRE é realizada via *pip*, incluindo todas as dependências opcionais necessárias para executar a *pipeline* completa:

```
>pip install "tigre[all]"
```

3.2 Execução da *pipeline*

A *pipeline* completa da TIGRE é executado através de três comandos sequenciais, todos em modo *multiple* para processamento paralelo dos 1.207 genomas.

3.2.1 Comando *clean*

O primeiro passo prepara e padroniza os arquivos GFF3:

```
>tigre clean multiple -v --log clean.log \
    --gdict plants_mit.gdict --tsv plants_mit.tsv
```

Este comando:

- Processa todos os genomas listados em `plants_mit.tsv`.
- Utiliza o arquivo .gdict em conjunto da ferramenta GDT, para padronizar nomes de genes.
- Resolve sobreposições e fragmenta *features* que atravessam limites circulares.
- Gera arquivos com sufixo `_clean`.
- Registra todas as operações em `clean.log`.

3.2.2 Comando *extract*

Com os arquivos preparados, o segundo passo extrai as regiões intergênicas:

```
>tigre extract multiple -v --log extract.log \
    --tsv plants_mit.tsv
```

Este comando:

- Lê os arquivos processados (sufixo `_clean`).
- Identifica intervalos entre *features* consecutivas.
- Cria anotações GFF3 das regiões intergênicas.
- Gera arquivos com sufixo `_intergenic`.
- Registra operações em `extract.log`.

3.2.3 Comando *getfasta*

O passo final recupera as sequências de nucleotídeos das regiões extraídas:

```
>tigre getfasta multiple -v --log getfasta.log \
    --tsv plants_mit.tsv --bedtools-compatible
```

Este comando:

- Lê as anotações intergênicas (sufixo `_intergenic`).
- Extrai sequências dos arquivos FASTA originais.

- Gera arquivos FASTA multi-sequência.
- Utiliza indexação base-0 nos cabeçalhos (compatível com *bedtools*).
- Registra operações em `getfasta.log`.

3.3 Desempenho e Requisitos de Hardware

As estratégias de otimização implementadas permitem sua execução em uma ampla variedade de configurações de *hardware*. Qualquer computador moderno (*desktop* ou laptop) com pelo menos 8 GB de RAM é capaz de utilizar a ferramenta para análise de milhares de genomas de organelas.

Para avaliar o desempenho da ferramenta, a *pipeline* completa foi executada sobre o *dataset* de 1.207 genomas mitocondriais de plantas em duas configurações distintas de *hardware*:

- **Servidor:** Ubuntu 22.04.5 LTS, dual Intel Xeon E5-2640 v4 @ 2.40 GHz (20 cores / 40 *threads*), 384 GB RAM
- **Laptop:** macOS Sonoma 14.6.1, Intel Core i9-8950HK @ 2.90 GHz (6 cores / 12 *threads*), 32 GB RAM

As métricas de desempenho foram coletadas utilizando o *GNU time* (`/usr/bin/time`) com a *flag* `-f` para formatação customizada, capturando tempo decorrido (`%E`) e consumo máximo de memória (`%M`). Os comandos foram os mesmos apresentados na subseção 3.2.1, subseção 3.2.2 e subseção 3.2.3.

```
/usr/bin/time -f "%E elapsed, %M KB max memory" <comando>
```

A Tabela 3 apresenta os tempos de execução e consumo máximo de memória para cada comando do pipeline nas duas configurações testadas.

Tabela 3 – Desempenho no processamento de 1.207 genomas mitocondriais.

Comando	Sistema Operacional	Tempo (s)	Memória (MB)
clean	Ubuntu 22.04.5 LTS	7.9	144.7
	macOS Sonoma 14.6.1	22.1	155.7
extract	Ubuntu 22.04.5 LTS	1.6	104.4
	macOS Sonoma 14.6.1	6.7	93.1
getfasta	Ubuntu 22.04.5 LTS	1.6	106.3
	macOS Sonoma 14.6.1	5.5	129.3

Fonte: Autoria Própria (2025).

Os resultados demonstram que a ferramenta TIGRE é capaz de processar mais de mil genomas em poucos segundos, mesmo em *hardware* modesto. O tempo total de execução da *pipeline* completa foi de 11.1 segundos no servidor e 34.3 segundos no laptop. O consumo

de memória permaneceu consistentemente baixo em ambas as configurações, com pico máximo de 155.7 MB, demonstrando a eficiência das otimizações implementadas, especialmente a arquitetura servidor-cliente para compartilhamento do arquivo .gdict durante o comando *clean*.

O comando *clean* apresentou o maior tempo de execução, refletindo a complexidade das operações de padronização, resolução de sobreposições e fragmentação de *features*. Os comandos *extract* e *getfasta* executaram em tempos significativamente menores, validando a estratégia de *design* onde o processamento intensivo é concentrado na etapa de preparação.

4 CONSIDERAÇÕES FINAIS

Este trabalho apresentou a TIGRE, uma ferramenta de linha de comando desenvolvida em *Python* para extração automatizada de regiões intergênicas a partir de genomas anotados. A ferramenta foi concebida inicialmente como uma etapa de pré-processamento em um projeto de pesquisa focado na análise de regiões intergênicas de organelas de plantas. No entanto, durante o desenvolvimento, identificou-se o potencial de generalização dessa funcionalidade, resultando na transformação de um *script* específico em uma ferramenta *standalone* de uso geral para comunidade científica.

A disponibilização da TIGRE como ferramenta *open source* sob licença MIT, publicada no PyPI e com código fonte no *GitHub*, possibilita o acesso a uma solução robusta para extração de regiões intergênicas. A aplicação prática demonstrada neste trabalho, processando 1.207 genomas mitocondriais em poucos segundos com consumo mínimo de memória, valida a eficácia da ferramenta em análises de larga escala.

O sucesso da ferramenta é evidenciado por sua adoção contínua no projeto de pesquisa original, onde já processou mais de 35.000 ANs de diferentes organismos, demonstrando sua robustez e aplicabilidade além do escopo inicial de genomas mitocondriais de plantas. A ferramenta e seu protocolo de uso estão em processo de publicação na revista *STAR Protocols*, facilitando sua adoção pela comunidade científica mais ampla.

A trajetória, de componente interno de pesquisa à ferramenta pública consolidada, ilustra como soluções desenvolvidas para problemas específicos podem, quando adequadamente generalizadas e documentadas, contribuir significativamente para a comunidade científica, facilitando pesquisas que dependem de processamento eficiente de dados genômicos em larga escala.

REFERÊNCIAS

COCK, P. J. A. *et al.* Biopython: freely available python tools for computational molecular biology and bioinformatics. **Bioinformatics**, v. 25, n. 11, p. 1422–1423, 03 2009. ISSN 1367-4803. Disponível em: https://doi.org/10.1093/bioinformatics/btp163.

DUPIN, B. *et al.* **Protocol for GDT, Gene Dictionary Tool, to create and implement a gene dictionary across annotated genomes**, . 2025. Disponível em: https://www.biorxiv.org/content/10.1101/2025.06.15.659783v1.

EDDY, S. R. The encode project: Missteps overshadowing a success. **Current Biology**, v. 23, n. 7, p. R259–R261, 2013. ISSN 0960-9822. Disponível em: https://www.sciencedirect.com/science/article/pii/S0960982213002893.

GOLDFARB, T. *et al.* Ncbi refseq: reference sequence standards through 25 years of curation and annotation. **Nucleic Acids Research**, v. 53, n. D1, p. D243–D257, 11 2024. ISSN 1362-4962. Disponível em: https://doi.org/10.1093/nar/gkae1038.

GUIGÓ, a. . S. R. Genome annotation: From human genetics to biodiversity genomics. **Cell Genomics**, v. 3, n. 8, p. 100375, 2023. ISSN 2666-979X. Disponível em: https://www.sciencedirect.com/science/article/pii/S2666979X23001726.

LEÓN, A.; PASTOR, Ó. Enhancing precision medicine: A big data-driven approach for the management of genomic data. **Big Data Research**, v. 26„ p. 100253, 2021. ISSN 2214-5796. Disponível em: https://www.sciencedirect.com/science/article/pii/S2214579621000708.

MATTICK, J. S. *et al.* Long non-coding rnas: definitions, functions, challenges and recommendations. **Nature Reviews Molecular Cell Biology**, v. 24, n. 6, p. 430–447, Jun 2023. ISSN 1471-0080. Disponível em: https://doi.org/10.1038/s41580-022-00566-8.

NELSON, C. E.; HERSH, B. M.; CARROLL, S. B. The regulatory content of intergenic dna shapes genome architecture. **Genome Biology**, v. 5, n. 4, p. R25, Mar 2004. ISSN 1474-760X. Disponível em: https://doi.org/10.1186/gb-2004-5-4-r25.

QUINLAN, A. R.; HALL, I. M. Bedtools: a flexible suite of utilities for comparing genomic features. **Bioinformatics**, v. 26, n. 6, p. 841–842, 01 2010. ISSN 1367-4803. Disponível em: https://doi.org/10.1093/bioinformatics/btq033.

Sanita Lima, M. *et al.* Pervasive transcription of plant organelle genomes: functional noncoding transcriptomes? **Trends in Plant Science**, v. 29, n. 6, p. 626–629, 2024. ISSN 1360-1385. Disponível em: https://www.sciencedirect.com/science/article/pii/S1360138524000190.

The Sequence Ontology. **GFF3 Specifications**, . 2024. Disponível em: https://github.com/The-SequenceOntology/Specifications/blob/master/gff3.md.