COPPE
Instituto Alberto Luiz Coimbra de Pós-Graduação e Pesquisa de Engenharia
UFRJ

BIOPROV: UMA BIBLIOTECA PARA DADOS DE PROVENIÊNCIA EM *WORKFLOWS* DE GENÔMICA COMPARATIVA

Vinícius Werneck Salazar

Dissertação de Mestrado apresentada ao Programa de Pós-graduação em Engenharia de Sistemas e Computação, COPPE, da Universidade Federal do Rio de Janeiro, como parte dos requisitos necessários à obtenção do título de Mestre em Engenharia de Sistemas e Computação.

Orientadores: Marta Lima de Queirós Mattoso
Fabiano Lopes Thompson

Rio de Janeiro

Março de 2021

BIOPROV: UMA BIBLIOTECA PARA DADOS DE PROVENIÊNCIA EM *WORKFLOWS* DE GENÔMICA COMPARATIVA

Vinícius Werneck Salazar

DISSERTAÇÃO SUBMETIDA AO CORPO DOCENTE DO INSTITUTO ALBERTO LUIZ COIMBRA DE PÓS-GRADUAÇÃO E PESQUISA DE ENGENHARIA DA UNIVERSIDADE FEDERAL DO RIO DE JANEIRO COMO PARTE DOS REQUISITOS NECESSÁRIOS PARA A OBTENÇÃO DO GRAU DE MESTRE EM CIÊNCIAS EM ENGENHARIA DE SISTEMAS E COMPUTAÇÃO.

Orientadores: Marta Lima de Queirós Mattoso
Fabiano Lopes Thompson

Aprovada por: Profa. Marta Lima de Queirós Mattoso
Prof. Fabiano Lopes Thompson
Prof. Daniel Cardoso Moraes de Oliveira
Prof. Alexandre de Assis Bento Lima
Prof. Diogo Antônio Tschoeke

RIO DE JANEIRO, RJ – BRASIL

MARÇO DE 2021

Salazar, Vinícius Werneck

BioProv: uma biblioteca para dados de proveniência em *workflows* de genômica comparativa/Vinícius Werneck Salazar. – Rio de Janeiro: UFRJ/COPPE, 2021.

XIV, 110 p.: il.; 29,7cm.

Orientadores: Marta Lima de Queirós Mattoso

Fabiano Lopes Thompson

Dissertação (mestrado) – UFRJ/COPPE/Programa de Engenharia de Sistemas e Computação, 2021.

Referências Bibliográficas: p. 79 – 89.

1. W3C-PROV. 2. Python científico. 3. PROV. 4. BioPython. 5. estruturas de dados. 6. pipelines. I. Mattoso, Marta Lima de Queirós *et al.* II. Universidade Federal do Rio de Janeiro, COPPE, Programa de Engenharia de Sistemas e Computação. III. Título.

Para Homero e Marylena.

Agradecimentos

Gostaria de agradecer aos meus orientadores, a profa. Marta Mattoso e o prof. Fabiano Thompson, por todo o aprendizado, pelas oportunidades e por serem exemplos de competência e dedicação. Agradeço aos professores da banca por terem aceitado o convite e por oferecem seu conhecimento e tempo para contribuir para esse trabalho. Prof. Daniel, obrigado pelas contribuições para a biblioteca BioProv; prof. Alexandre, obrigado pelos ensinamentos nas disciplinas do PESC; prof. Diogo, obrigado pelas colaborações, pelas disciplinas e também por cuidar da nosso valioso servidor.

Agradeço ao PESC e o CNPq por cederem generosamente a bolsa de estudos que financiou esse mestrado, e à secretaria do PESC: Ricardo, Gutierrez, Mercedes, Solange e muitos outros por serem sempre cordiais e prestativos. Gostaria de agradecer à dra. Kary Ocaña e às mestras Débora Pina e Liliane Neves pelos aprendizados na área da computação. Aos colegas do Laboratório de Microbiologia, que são muitos para listar: prof. Jean Swings, Tatiane, Tooba, Gustavo, Hannah, Michele, Pedro, Bruno Masi, Bruno Sérgio, Raphael, Larissa, Mayanne, Bruna, Isabel, Angélica, Itamar e muitas outras pessoas. Agradeço aos amigos do Rio e o grupo do Apê Dry-Wall por nos acolherem e se tornarem amizades que vão perdurar pelo resto da vida.

Agradeço também aos grupos de conversa, como o Brasileiros na SciPy e o Código Bonito. Agradeço aos colegas Tiago Lubiana e João Vitor Cavalcante, que acabou tornando-se o primeiro contribuidor da BioProv. Agradeço à toda a comunidade open source por todo o trabalho que vocês fazem.

Queria fazer um agradecimento especial para a minha família, principalmente aos meus pais, Luciana e Renato, pelo apoio e amor incondicional. Aos meus irmãos, Marina e Diego, por serem exemplos de pessoa. Por último, queria agradecer minha companheira, Thayana, pela parceria, pelo apoio, pelos batepapos científicos e por estar junto de mim em todos os momentos. Me traz orgulho e alegria também poder acompanhar a sua formação como cientista, e todo dia eu aprendo algo novo com você.

Resumo da Dissertação apresentada à COPPE/UFRJ como parte dos requisitos necessários para a obtenção do grau de Mestre em Ciências (M.Sc.)

BIOPROV: UMA BIBLIOTECA PARA DADOS DE PROVENIÊNCIA EM *WORKFLOWS* DE GENÔMICA COMPARATIVA

Vinícius Werneck Salazar

Março/2021

Orientadores: Marta Lima de Queirós Mattoso
Fabiano Lopes Thompson

Programa: Engenharia de Sistemas e Computação

Em análises computacionais, a proveniência de um experimento caracteriza como, quando, por que e por quem um este experimento foi executado. Na bioinformática, a captura e análise de dados de proveniência ainda é um desafio, devido à complexidade dos *workflows*, heterogeneidade dos dados biológicos e a ausência de bibliotecas especializadas. As soluções existentes não possuem funcionalidades para captura de proveniência em um formato estruturado, e/ou não apoiam as especificidades do domínio da bioinformática. Esta dissertação apresenta BioProv, uma biblioteca de *software* para extração, representação e análise de dados de proveniência de *workflows* de bioinformática, com compatibilidade com a recomendação W3C-PROV. A biblioteca é utilizada em um estudo de caso de um *workflow* de taxonomia genômica. Nesse estudo, foi possível capturar a proveniência, extrair dados específicos de domínio, e armazená-los em disco com um adicional de no máximo 3.56% de tempo de execução. Tais resultados destacam a capacidade de BioProv de extrair, armazenar e consultar dados de proveniência com uma sobrecarga computacional desprezível.

Abstract of Dissertation presented to COPPE/UFRJ as a partial fulfillment of the requirements for the degree of Master of Science (M.Sc.)

BIOPROV: A LIBRARY FOR PROVENANCE DATA IN COMPARATIVE GENOMICS WORKFLOWS

Vinícius Werneck Salazar

March/2021

Advisors: Marta Lima de Queirós Mattoso
Fabiano Lopes Thompson

Department: Systems Engineering and Computer Science

In computational analyses, the provenance of an experiment characterizes how, when, why, and by whom the experiment was performed. In bioinformatics, monitoring provenance is still a challenge, due to the complexity of workflows, the heterogeneity of biological data and the absence of specialized libraries. Existing solutions do not possess features for provenance capture in a structured format, and/or do not support the specificities of the bioinformatics' domain. This dissertation presents BioProv, a software library for the extraction, representation and analysis of provenance data of bioinformatics workflows, with compatibility with the W3C-PROV standard. The library is utilized in a case study of a genomic taxonomy workflow. In this study, it was possible to capture provenance, extract domain data, and store them on disk with a maximum increase of 3.56% in the execution time. Such results highlight the capacity of BioProv of extracting and storing provenance data with negligible computational overhead.

Sumário

Lista de Figuras

Lista de Tabelas

Lista de Abreviaturas

AAI	*average amino acid identity*[2], p. 55
ANI	*average nucleotide identity*[2], p. 55
API	*application-programming interface*, p. 11
CLI	*command-line interface*, ou interface de linha de comando, p. 24
CWL	Common Workflow Language, p. 20
DAG	*directed acyclic graph*, ou grafo acíclico direcionado, p. 20
DfA	DfAnalyzer, p. 21
HTS	*High-throughput sequencing*, ou sequenciamento de alto desempenho., p. 5
IO	Input/Output, ou entrada e saída, p. 37
LNCC	Laboratório Nacional de Computação Científica, p. 16
PyPI	Python Package Index, o repositório de pacotes da Python Software Foundation, p. 34
SGBD	Sistema de gerência de banco de dados, p. 24
SGWF	Sistema de gerência de *workflows* científicos, p. 8
W3C	*World Wide Web Consortium*, p. 7
WFB	*Workflow* de bioinformática, p. 6

Lista de Snippets de código

Capítulo 1

Introdução

A bioinformática é uma ciência, por definição, interdisciplinar, que une princípios da biologia e da computação [1]. Essa dissertação, por consequência, também é um esforço interdisciplinar, e que visa apresentar uma solução para captura de proveniência que seja especializada para *workflows* bioinformáticos. Esta é uma questão crítica em ciências computacionais, incluindo a bioinformática, pois a captura de proveniência é um aspecto essencial para gerenciar as diferentes etapas do ciclo-de-vida de um experimento [2]. A definição de proveniência utilizada nessa dissertação é a proposta pelo grupo de trabalho de proveniência 'PROV' do *World Wide Web Consortium*, que declara que:

> "Proveniência é informação sobre entidades, atividades e pessoas envolvidas na produção de um dado ou coisa, que pode ser usada para informar avaliações sobre a sua qualidade, credibilidade e confiabilidade" [3].

Essa definição foi criada especialmente para o compartilhamento de informações de proveniência entre diferentes sistemas e ambientes, e se mostra útil para uma variedade de aplicações para além da pesquisa científica, como jornalismo de dados, contabilidade, processos de produção, e virtualmente qualquer outra aplicação que envolva o fluxo de dados através de processos e associados a agentes [4]. Como a bioinformática é uma ciência emergente, guiada por dados, e que passou por profundas transformações nos últimos anos, o monitoramento e a captura de proveniência se tornam aspectos importantes das práticas de pesquisa dessa disciplina, assim como em outras áreas de conhecimento que foram revolucionadas por análises computacionais; apesar de uma notável difusão de boas práticas de computação científica entre a comunidade da bioinformática em anos recentes [5], realizar "*provenance-aware research*" ainda é um desafio para muitos grupos de pesquisa [6].

Na bioinformática, um fato notável que contribui para esse desafio é a ausência de bibliotecas de proveniência que sejam especializadas para este domínio de aplicação, ou seja, que contemplem os tipos de dados e programas usados em análises de bioinformática. Isto se torna um problema pois mesmo que o usuário consiga capturar a proveniência de uma determinada análise computacional, a ausência de um formato estruturado e interoperável impede que o usuário seja capaz de consultar esses dados de forma analítica. A consequência disso é uma dificuldade para interpretar como os parâmetros do experimento afetaram os resultados obtidos, bem como compreender as relações entre diferentes arquivos e programas na análise, como quais parâmetros foram utilizados, quais arquivos foram consumidos ou gerados por cada programa, ou mesmo qual usuário foi responsável pela execução de um programa ou criação de um determinado arquivo.

As soluções existentes para captura e análise de proveniência contribuem para a resolução desse problema, porém como será detalhado mais adiante, tais soluções: i) não são capazes de capturar proveniência em um formato estruturado e interoperável; ou ii) são capazes de capturar proveniência em um formato estruturado, porém não possuem funcionalidades especializadas para bioinformática (são soluções "genéricas" ou especializadas em outro domínio). No primeiro caso, mesmo que o usuário obtenha algum tipo de proveniência através de, por exemplo, *logs*, relatórios e/ou traços de execução, a ausência de um formato estruturado e interoperável limita as consultas analíticas aos resultados. No segundo caso, apesar da possibilidade do usuário de realizar análises sobre os dados de proveniência, essas análises tornam-se superficiais, pois como as soluções são genéricas (não são especializadas em bioinformática), os dados de domínio relevantes para *workflows* de bioinformática.

Logo, o problema principal abordado nessa dissertação é a ausência de bibliotecas de captura de proveniência especializadas em bioinformática, e a consequente dificuldade para extração e análise de dados proveniência em workflows de bioinformática. A solução proposta é uma biblioteca de software para captura de proveniência em workflows de bioinformática, com uma representação de classes para os elementos de proveniência, e que siga um formato estruturado e interoperável.

O objetivo dessa dissertação, portanto, é apresentar uma solução que facilite a captura e análise de dados de proveniência por parte do(a) usuário(a) e que seja familiar para a pesquisa em bioinformática. Para isso, foi desenvolvida BioProv, uma biblioteca para captura de dados e geração de documentos de proveniência, e que é especializada em *workflows* bioinformáticos. BioProv funciona como um

wrapper[1] para a execução de programas na linha de comando do sistema, capturando os dados de proveniência da execução, e fornece um esquema de representação de proveniência através de classes de alto nível. As funcionalidades de captura de proveniência da biblioteca permitem a extração automática de dados de domínio, através da disponibilização de *parsers* para formatos de arquivo de dados biológicos e a programas amplamente utilizados na bioinformática. Uma vez que esses dados de proveniência são coletados, podem ser inseridos no banco de dados interno da aplicação, exportados em formato JSON ou usados para a criação de documentos de proveniência compatíveis com o formato W3C-PROV [3], e que podem ser serializados em diversos formatos.

Para uma melhor compreensão da contribuição feita por essa dissertação, o texto a seguir é dividido em quatro capítulos capítulos.

O Capítulo 2 contextualiza os temas principais da dissertação. A transformação das ciências biológicas por conta do surgimento das tecnologias de sequenciamento de alto desempenho é um fenômeno extensivamente documentado na literatura, e uma discussão igualmente importante é o papel das análises computacionais no advento científico. Está estabelecido que um aspecto fundamental da pesquisa computacional, e que serve para reforçar a reprodutibilidade dos experimentos, é o monitoramento de proveniência. Nesse capítulo, é feita uma introdução do campo da bioinformática, e de como o fator limitante para produção de conhecimento deixou de ser a capacidade de geração de dados para se tornar a capacidade de processamentos dos mesmos. Essa mudança impulsionou a necessidade de reforçar uma série de princípios para melhorar a reprodutibilidade dos experimentos e as práticas de gestão de dados. Um desses princípios é o monitoramento ativo de proveniência em todos os ciclos de vida do experimento. No entanto, esse é um desafio no campo da bioinformática por conta das características dos *workflows* e também pela escassez de ferramentas adequadas para os usuários. São apresentadas algumas tecnologias e trabalhos relacionados que são relevantes para o tema de proveniência em *workflows* bioinformáticos, incluindo a recomendação W3C-PROV, sistemas de gerência de *workflows* científicos, e pacotes para realizar captura de proveniência ou a extração de dados de domínio.

O Capítulo 3 apresenta a contribuição principal dessa dissertação, a biblioteca BioProv. Uma visão geral é descrita, incluindo a arquitetura de *software*, exemplos básicos de uso e os objetivos propostos. Em seguida, são apresentados os detalhes de implementação, que incluem como a biblioteca é dividida em

[1] Um *wrapper* é uma função ou serviço de software cuja função é chamar uma função ou rotina secundária com pouca ou nenhuma computação adicional.

diferentes subpacotes e módulos, uma descrição de cada classe implementada, como é feita a compatibilidade da biblioteca com o modelo W3C-PROV e como as dependências de BioProv são utilizadas. BioProv é uma biblioteca de código aberto, e nesse capítulo também são discriminadas quais foram as contribuições feitas por terceiros para a versão v0.1.22 da biblioteca.

No Capítulo 4, é feita uma avaliação experimental da biblioteca. Para isso, um workflow de taxonomia genômica foi adaptado de dois estudos desenvolvidos durante a produção dessa dissertação. O *workflow* foi instrumentado em um script Python e executado para conjuntos de dados contendo dez, cinquenta e cem amostras, e a sobrecarga computacional (acréscimo no tempo de execução em segundos) foi medida com e sem as chamadas para a biblioteca BioProv. Também são apresentadas consultas ao banco de dados da aplicação, que ilustram os dados de proveniência coletados. Esses resultados demonstram uma implementação concreta da biblioteca, e a sua eficácia e limitações para a coleta de proveniência em *workflows* de bioinformática.

Ao fim de cada um desses capítulos, é apresentado um sumário que resume as ideias abordadas no capítulo. Por último, o Capítulo 5 faz uma síntese do texto principal, onde são revisitados os temas dos capítulos anteriores. Também são delineadas as direções futuras da biblioteca BioProv, que destaca novas ideias de funcionalidades que poderiam inspirar esforços futuros.

Os apêndices apresentam os tutoriais de uso, que fazem parte da documentação da biblioteca, e o resumo e pôster aceitos para apresentação no X-Meeting 2019, que discutem o papel da proveniência de dados na gestão do conhecimento em bioinformática. Esses anexos, portanto, constituem partes integrais dessa dissertação.

Capítulo 2

Captura de proveniência e workflows de bioinformática

Duas ou três décadas atrás, o lugar mais comum para se encontrar um biólogo seria atrás da lente de um microscópio ou sobre uma bancada, cuidadosamente pipetando líquidos em tubos de vidro. Hoje em dia, grande parte destes profissionais se encontram em escritórios, olhando para linhas de código e analisando planilhas com grandes volumes de dados [7]. O uso de técnicas computacionais se tornou tão central no entendimento da vida que pode-se argumentar que, atualmente, toda biologia é biologia computacional [1]. Essa "transformação digital" da prática de pesquisa em ciências da vida não está associada tão somente ao sentido mais amplo do termo, que se refere a digitalização de processos ao longo de diversos setores da sociedade, mas sim aos avanços específicos do campo, que verdadeiramente trouxeram "uma nova biologia para um novo século" [8].

No início dos anos 2000, a consolidação da tecnologia de sequenciamento de alto desempenho (*high-throughput sequencing*, HTS) revolucionou as ciências biológicas [9]. O aumento vertiginoso na produção de dados de sequências biológicas (Figura 2.1), coordenado através de iniciativas globais como a *International Nucleotide Sequence Database Collaboration*, *Human Genome Project* e o *Earth Microbiome Project*, permitiu avanços extraordinários em áreas amplas como agricultura, biomedicina, biotecnologia, e ciências ambientais [10, 11]. A limitação nessa área da pesquisa científica deixou de ser a capacidade de **geração de dados**, passando a se tornar a capacidade para **armazenamento, processamento e análise dos dados**. Esse "gargalo da bioinformática"[12, 13] passou a ser tanto uma oportunidade quanto um desafio para cientistas, na busca por armazenar e consultar esses dados através de serviços públicos, e obter o máximo de informação dos dados que lhes estavam disponíveis. A biologia se tornou uma ciência de

big data, acompanhada de todas as possibilidades e problemas que isso implica [7, 14, 15].

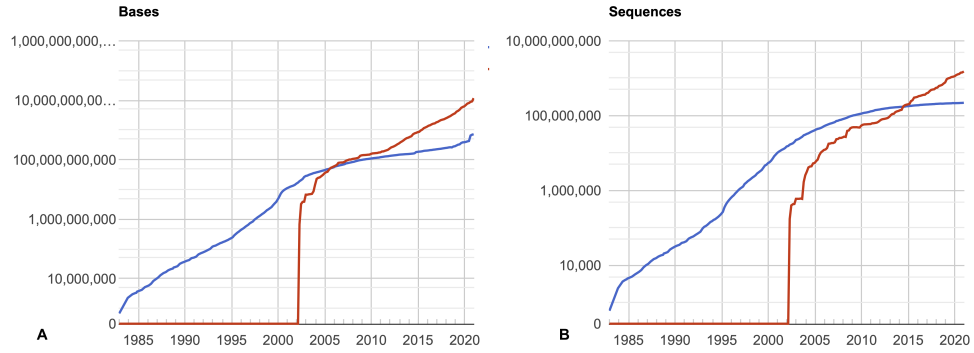


Figura 2.1: Aumento no número de sequências nos bancos de dados NCBI GenBank e WGS. O painel A mostra o crescimento no número de bases ao longo dos anos e o painel B mostra o crescimento no número de sequências. Em azul: banco NCBI GenBank. Em vermelho: banco WGS. Fonte: https://www.ncbi.nlm.nih.gov/genbank/statistics/

Consequentemente, análises e protocolos que realizam o processamento de sequências biológicas crescem tanto em diversidade quanto em complexidade. O desenvolvimento da *e-Science*[1] para apoiar a pesquisa em biologia computacional é fundamental para reforçar a robustez e reprodutibilidade dos resultados obtidos através do processamento computacional com ***workflows* de bioinformática** (WFBs). Workflows científicos são definidos como "um conjunto de tarefas computacionais e de tratamento de dados que são interrelacionadas e visam cumprir um determinado objetivo" [17]; os *workflows* científicos na área da bioinformática têm suas próprias particularidades e características comuns que são discutidas na Seção a seguir. Embora o foco dessa dissertação sejam WFBs, é importante enxergá-los dentro do contexto da *e-Science* como um todo.

Um aspecto importante do método científico, cujo a *e-Science* busca trazer para a experimentação *in silico*, é o registro da **proveniência** de um determinado experimento; descrever detalhadamente o que foi feito, quando, porque e por quem não é somente recomendado, mas é absolutamente necessário para aplicar o rigor científico em análises computacionais [6, 18]. O suporte à proveniência não só melhora a qualidade de tais análises, mas é uma condição necessária para a gestão do conhecimento em bioinformática, ou seja, para que o conhecimento acerca de um processo ou análise possa ser transferido, armazenado e compartilhado [19]. Proveniência, no contexto de pesquisa computacional, pode ser des-

[1] *e-Science* é um termo usado para se referir a ciência colaborativa e global, realizada *in silico* com uma determinada infraestrutura computacional [12, 16].

crita concisamente como "a descrição da origem de um determinado de dado, por qual(is) processo(s) ele foi produzido e como chegou no banco de dados" [20]. Uma definição mais explícita de proveniência é a do Grupo de Trabalho de Proveniência 'PROV' do *World Wide Web Consortium* (W3C), que a define como "um registro que descreve pessoas, instituições, entidades, e atividades envolvidas em produzir, influenciar, ou entregar um determinado dado ou coisa." [4]

Aplicar práticas de proveniência em estudos de bioinformática é um aspecto importante para assegurar a reprodutibilidade dos experimentos. O conceito de uma "crise de reprodutibilidade" é amplamente discutido em diversas áreas da pesquisa científica: isso não é um problema exclusivo da bioinformática. A ubiquidade de dados tem como consequência uma torrente de resultados que muitas vezes falham em ser reproduzidos; embora esse aspecto tenha melhorado bastante nos anos recentes, continua como um desafio para a biologia computacional e outros campos científicos [21–23].

Esforços para melhorar a reprodutibilidade na bioinformática e áreas relacionadas são direcionados principalmente às atividades de deposição e consulta de dados, à disponibilidade de *software* e código utilizado para as análises, e ao uso de padrões de dados interoperáveis [21]. Nesse contexto, podemos identificar três conjuntos de princípios que reforçam a condução dessas atividades:

- os princípios FAIR para administração de dados [24];
- a aplicação de boas práticas de computação científica [5, 25];
- o monitoramento ativo de proveniência em experimentos computacionais [2, 26].

O primeiro conjunto se refere aos princípios FAIR: *findable, accessible, interoperable* e *reusable*. O movimento FAIR almeja que todos os objetos de pesquisa devem ser encontráveis, acessíveis, interoperáveis e reutilizáveis, tanto por humanos quanto por máquinas [24]. Os conceitos de "objetos reutilizáveis de pesquisa" [27] e *data FAIRness* se referem respectivamente aos artefatos computacionais produzidos por um estudo científico, sejam eles códigos, dados, figuras ou resultados, e a capacidade de aumentar a qualidade e disponibilidade desses objetos para ajudar na descoberta de conhecimento. É importante notar que os princípios FAIR são relacionados, mas são separáveis e independentes; dados altamente sensíveis, como por exemplo dados médicos, talvez possuam mecanismos para serem facilmente encontráveis, interoperáveis e reutilizáveis, mas definam regras claras a respeito de sua acessibilidade. Logo, tais princípios podem ser aplicados incrementalmente para promover *FAIRness* de dados [24]. Na

bioinformática, especificamente nas subáreas relacionadas à genômica ou a análise de sequências biológicas, os princípios FAIR também se relacionam com os padrões particulares de deposição de dados. O estabelecimento de "informações mínimas sobre uma sequência" [28] e padrões relacionados [29–31], promovidos pelo *Genomics Standards Consortium*[2], garantem que dados de sequências biológicas tenham o mínimo de informação necessária para a integração de dados, para estudos comparativos e para a descoberta de conhecimento.

O segundo conjunto de princípios trata principalmente da aplicação de boas práticas metodológicas em projetos computacionais; é reconhecido que essa é uma habilidade essencial para desenvolver ciência de qualidade, e também que existe uma lacuna de treinamento dessas habilidades na formação de muitos cientistas [5]. Atualmente, é comum para cientistas de áreas diversas passarem boa parte do seu tempo desenvolvendo *software* ou scripts para análise de seus dados. No entanto, com a exceção das ciências informáticas, matemáticas e da computação, e de alguns ramos das engenharias, raramente esses cientistas têm treinamento formal em boas práticas computacionais [25, 32]. Esse cenário está melhorando [33, 34], mas ainda existe um desequilíbrio evidente entre o quanto esse treinamento beneficia esses cientistas e o quanto ele é oferecido durante sua formação. Boas práticas de computação científica não beneficiam somente os cientistas que as adotam de forma individual, mas também facilitam a colaboração entre grupos de pesquisa e em disciplinas como um todo. Essas práticas envolvem diretrizes para o desenvolvimento e implementação de *software*, o uso de ferramentas adequadas para as tarefas computacionais requeridas, e a documentação metodológica tanto do código quanto do projeto de pesquisa em si [5].

O terceiro conjunto de princípios, sobre o monitoramento de proveniência, propõe a adoção de uma prática de pesquisa guiada por proveniência não somente para a execução e análises dos resultados de experimentos, mas também para a composição e planejamento dos mesmos. A coleta de proveniência em todas as fases do ciclo de vida de um experimento computacional representa um desafio de pesquisa, e adequar as particularidades metodológicas de cada *workflow* para isso pode ser extremamente custoso, se não for feito com o suporte das ferramentas propícias para tal [2]. Uma categoria de *software* amplamente empregada em pesquisa computacional são sistemas de gerência de *workflows* científicos (SGWFs); tais sistemas são usados para orquestração e execução de workflows científicos, e permitem o armazenamento, consulta e visualização de dados de execução. No entanto, esses sistemas usualmente contemplam somente cada *execução isolada de um workflow*, tratando-as forma desconectada no contexto

[2] https://gensc.org/

do experimento computacional como um todo. Uma abordagem mais compreensiva para a prática de pesquisa computacional envolve a coleta de proveniência em todos os estágios do ciclo de vida do experimento [2]. Essa abordagem se traduz em uma série de desafios práticos, alguns particulares para WFBs [18, 19, 35], pois estão relacionados as especificidades do campo da bioinformática em termos de conteúdo e estruturação de ambos os dados e as análises. A coleta de dados de proveniência proporciona uma concepção mais nítida da composição do experimento, um controle mais preciso durante a sua execução, e uma análise de resultados mais detalhada. Experimentos cuja proveniência é monitorada podem ser ajustados durante o tempo de execução de acordo com os dados que são coletados, logo tarefas subsequentes podem ser informadas por dados de proveniência.

Esses três conjuntos de princípios, embora independentes, estão relacionados e atuam em sinergia para melhorar a reprodutibilidade da pesquisa computacional, e por consequência contribuir para a integridade do conhecimento científico resultante. O conceito de proveniência de dados, embora destacado no último conjunto, é comum aos três conjuntos de princípios, pois é um aspecto essencial da prática de pesquisa computacional, independente da área de domínio ao qual esta é aplicada. Logo, é o conceito unificador dessa dissertação, que busca fazer uma contribuição para a pesquisa em proveniência no campo da bioinformática.

A próxima Seção descreve características comuns e particulares de WFBs; em seguida, serão delineados alguns desafios específicos para a coleta de proveniência nesse campo. Por último, serão apresentadas tecnologias e trabalhos relacionados que fornecem uma base teórica e prática para o conteúdo dos capítulos subsequentes.

2.1 Características de WFBs

Bioinformática é, por definição, uma ciência interdisciplinar que utiliza técnicas computacionais para a análise, processamento e armazenamento de dados biológicos [36]. Essa é uma definição intencionalmente ampla. No entanto, desde a sua concepção na década de 1950 com os trabalhos de Margaret Dayhoff, passando pelas inovações em sequenciamento realizados por Frederick Sanger na década de 1970 até a consolidação das plataformas HTS no início da década de 2000, a bioinformática sempre teve como o seu foco principal o estudo de **sequências biológicas**, especificamente sequências de nucleotídeos ou aminoácidos [37]. Apesar de existirem numerosas linhas de pesquisa no campo da bioinformática que uti-

lizam fontes de dados como sinais, imagens e linguagem natural, é inegável que as sequências biológicas são as entidades predominantes nesse campo. Pode-se até entender, de forma subjetiva, que grande parte da bioinformática na verdade é "biologia molecular computacional", pois é dedicada a análise das moléculas representadas em sequências biológicas.

Apesar dessa aparente especialização em um domínio específico de dados, mesmo quando se trata apenas de sequências biológicas, a bioinformática ainda lida com uma enorme heterogeneidade de dados. Sequências vem em todas as formas e tamanhos: *short reads*, *long reads*, *contigs*, *scaffolds* e montagens a nível cromossomal são somente algumas das representações de moléculas de ácido nucleico em arquivos digitais. Esses arquivos também podem estar representados em diversos formatos, como FASTA, FASTQ, SAM, VCF, GFF, e inúmeros outros. Estes formatos são todos representações de texto plano (ASCII), no entanto, também existem formatos binários para bioinformática como BAM, ABI, e HDF5 (formato BioHDF). Os formatos FASTA e FASTQ [3] talvez sejam os formatos mais ubíquos para sequências biológicas, dada a sua simplicidade e versatilidade.

Além da quantidade de arquivos representando sequências, pesquisadores de bioinformática desenvolveram uma variedade de algoritmos para a análise das mesmas. Exemplos notórios incluem o algoritmo Needleman-Wunsch [38] e o algoritmo Smith-Waterman [39], para alinhamento. Métodos preexistentes, como a Transformada Burrows-Wheeler para compressão de dados [40], e Modelos Ocultos de Markov [41] também foram adaptados para o uso na bioinformática. Esses algoritmos para identificação, alinhamento e comparação de sequências se tornaram tão difundidos que a publicação original do *software* BLAST [42], que realiza busca por alinhamento de uma sequência contra um banco de dados de referência, é um dos artigos mais citados da ciência moderna[4], tamanha a sua usabilidade. Outro *software* proeminente é o Clustal [43], que realiza o alinhamento múltiplo de sequências. Os problemas de análise de sequência em bioinformática podem ser divididos de forma geral em quatro partes: alinhamento pareado (como BLAST), alinhamento múltiplo (como o Clustal), construção de árvores filogenéticas, e estrutura de RNA [41]. Cada uma dessas classes de problemas apresenta inúmeros programas que implementam algoritmos altamente especializados para atingir um determinado objetivo [44].

Diante dessa diversidade de ambos dados e algoritmos, WFBs raramente es-

[3] https://www.ncbi.nlm.nih.gov/BLAST/fasta.shtml

[4] Quando o número de citações para a publicação original, de 1990, é somada a da segunda versão, em 1997, o BLAST torna-se o *software* mais citado na história da bioinformática. Acompanhar a discussão em https://www.biostars.org/p/117390/ e https://www.nature.com/news/the-top-100-papers-1.16224.

tão limitados a um ou dois passos isolados. Uma única amostra biológica pode gerar um número grande de arquivos de sequências, que serão processados, filtrados, alinhados, e traduzidos para outros formatos. Um exemplo comum é a análise de comunidades microbianas através de genes marcadores, como o rRNA 16S. Um *workflow* típico com esse fim incluiria passos para controle de qualidade, aparamento de sequências, filtragem de erros, determinação taxonômica dos organismos, e cálculo de índices de diversidade [45]. Cada um desses passos processa as sequências de forma distinta, requer uma variedade de parâmetros que influenciam o resultado final, e cria múltiplos arquivos de saída com conteúdos diferentes mas que se referem a entidades (amostras biológicas) comuns. Diferentes tipos de análise também sofrem limitações estatísticas particulares; análises de comunidades por genes marcadores devem ser interpretadas conforme a natureza composicional dos dados obtidos[5]. Outros protocolos, tais como a reconstrução de genomas a partir de metagenomas, também apresentam seus próprios vieses, que devem ser considerados durante a etapa de interpretação dos resultados e de aplicação de testes estatísticos [48, 49].

O panorama atual do desenvolvimento de *workflows* de bioinformática, em um contexto de engenharia de *software*, é impulsionado pela utilização de sistemas de *workflow* que comportem as necessidades intensivas de processamento de dados destes *workflows* [50]. Portanto, somente através da instrumentação adequada de um *workflow* que é possível monitorar todas as informações pertinentes à sua execução. Isso inclui, além das próprias saídas do workflow, os dados de proveniência, como o relatório e grafo de execução e os logs. No entanto, nem todos os sistemas de gerência de *workflow* científico possuem funcionalidades que permitam a extração estruturada desses dados, e os que possuem apresentam abordagens particulares de como capturar e gerenciar os dados de proveniência [26].

Tornar um experimento computacional "*provenance-aware*" é, sem dúvida, uma prática fundamental em uma ciência cada vez mais guiada por *software* e dados digitais. No entanto, é interessante que sistemas capazes disso sejam projetados para acomodar as necessidades específicas de WFBs. Isso inclui um suporte a arquivos de sequência de variados formatos, reconhecimento de operações e problemas comuns no domínio da bioinformática, e uma API intuitiva para usuários que em sua maioria serão bioinformatas e não engenheiros de dados ou especialistas em proveniência. Claramente isso é um desafio de pesquisa; é comum que as plataformas existentes apresentem problemas como sobrecarga excessiva, dificuldade de instrumentação, ou não são capazes de coletar dados de domínio.

[5]Para uma discussão detalhada, referir a [46, 47].

No entanto, muito progresso já foi feito e a capacidade de tornar experimentos *provenance-aware* é cada vez mais acessível. Na Seção seguinte, serão apresentados alguns casos de captura de proveniência em WFBs, acompanhados dos principais problemas nesse tópico.

2.2 Desafios da captura de proveniência em WFBs

A capacidade de reproduzir um experimento é um dos pilares fundamentais da ciência. A reprodutibilidade dos resultados é algo que torna as descobertas científicas mais transparentes e robustas, e afeta diretamente a credibilidade do conhecimento científico. É evidente que apesar dos crescentes esforços para tornar experimentos computacionais mais reprodutíveis, ainda assim se vive uma crise de reprodutibilidade [23, 51]. Uma das soluções mais diretas para esse problema é priorizar práticas de proveniência na pesquisa computacional. Ao coletar dados de proveniência em todas as fases do ciclo de vida de um experimento, cientistas têm uma visão muito mais informada do mesmo, e portanto podem gerenciar sua pesquisa mais efetivamente; isso é especialmente verdadeiro para experimentos computacionais de larga escala [2].

O conceito de proveniência vem se tornando cada vez mais notável em várias disciplinas, especialmente as que passaram por grandes transformações devido aos avanços em geração em processamento de dados, como a biologia e a astronomia [6, 7]. No campo da bioinformática, embora práticas de proveniência tenham sido inicialmente negligenciadas ou mesmo rejeitadas por cientistas, é inegável que a sua crescente importância as tornaram um requerimento mínimo de boas práticas [5, 19].

As estratégias para capturar dados de proveniência em WFBs são diversas, e estão associadas ao ambiente computacional e plataforma de execução nas quais o experimento é realizado [26]. Essa associação implica em desafios, pois muitas vezes os SGWFs não suportam a captura de proveniência de uma forma satisfatória, com cada um desses sistemas apresentando suas vantagens e limitações particulares [52]. Outra questão é a heterogeneidade dos dados biológicos e os algoritmos especializados usados na análise de sequências. Idealmente, sistemas de proveniência para bioinformática devem ser capazes de extrair dados específicos de domínio, que por sua vez são úteis para direcionar os parâmetros do *workflow* [35].

Tais sistemas já foram implementados em WFBs que desempenham uma variedade de funções, como modelagem de homologias [53], descoberta de fármacos [54], filogenômica [55], e chamada de variantes [52], e em ambientes distintos,

como serviços de nuvens e clusters de alto desempenho. Dados de proveniência capturados se mostraram úteis para identificar falhas durante a execução, para o direcionamento de parâmetros, e para melhorar a reprodutibilidade dos experimentos [56]. No entanto, uma dificuldade persistente é a decisão de como modelar os dados de proveniência extraídos de um WFB; muitas vezes isso é realizado com colaborações entre os cientistas de domínio e especialistas em proveniência, mas essa dependência impede uma adoção mais ampla de práticas de proveniência pela comunidade da bioinformática. Para que a visão de uma prática de pesquisa baseada em proveniência seja concretizada, é necessário o estabelecimento de padrões interoperáveis e cuja implementação seja simples o suficiente para que usuários sejam capazes de incorporá-los em suas rotinas de trabalho, de forma de amenizar a transição cultural que isso implica [6]. Logo, pode-se destacar que um dos principais problemas para aplicar práticas de proveniência na pesquisa em bioinformática é a **ausência de bibliotecas especializadas que façam a captura, representação e análise de dados de proveniência**. Como citado no Capítulo 1, "as soluções existentes: i) não são capazes de capturar proveniência em um formato estruturado e interoperável; ou ii) são capazes de capturar proveniência em um formato estruturado, porém não possuem funcionalidades especializadas para bioinformática (são soluções "genéricas" ou especializadas em outro domínio)." O objetivo dessa dissertação, portanto, é contribuir para a resolução desse problema, propondo uma solução através de uma biblioteca que realize a captura e extração dos dados de proveniência, e que permita que o usuário realize consultas analíticas sobre os dados capturados, algo que é logrado pela utilização de um formato estruturado e interoperável.

Em sumário, é possível delinear certos fatores que são particularmente significativos para a aplicação de práticas de proveniência no campo da bioinformática. Estes incluem:

- A integração entre ferramentas de captura de proveniência e sistemas de gerência de *workflows* científicos, independente da plataforma computacional (ambiente local, nuvem, ou cluster de alto desempenho);
- A capacidade destas ferramentas de extraírem dados específicos de domínio, especialmente dados de sequências biológicas, e também o suporte a operações bioinformáticas comuns;
- A consolidação de padrões interoperáveis para representação de proveniência.

Para propor novas soluções que auxiliem a captura de proveniência, é necessário contextualizar as tecnologias existentes que são relacionadas a essa tarefa.

A seguir, serão apresentados alguns dos temas que são importantes na área da bioinformática e na pesquisa computacional de forma geral.

2.3 Tecnologias e trabalhos relacionados

A principal contribuição dessa dissertação é uma biblioteca para criação de documentos de proveniência para *workflows* de bioinformática, apresentada no Capítulo 3. Para uma compreensão adequada do escopo da biblioteca, é necessário descrever três tópicos relativos aos desenvolvimentos recentes na área de pesquisa em proveniência. Primeiramente, será apresentada a recomendação W3C-PROV para representação de proveniência. Os documentos exportados pela biblioteca BioProv são formatados conforme as especificações do W3C-PROV Data Model (W3C-PROV-DM), incluindo a representação dos diferentes elementos (entidade, atividade, e agente) e relações de proveniência. Em seguida, será feita a descrição de alguns sistemas de gerência de *workflow* científico, que são uma classe de *software* que se tornou essencial para a execução de *workflows* científicos, apesar das devidas limitações no aspecto de captura de proveniência. Por último, serão apresentados algumas bibliotecas relacionadas que apresentam propósitos similares ao da biblioteca BioProv.

2.3.1 W3C-PROV

Foram quase quatro anos após a primeira comunicação bem sucedida entre um cliente HTTP e um servidor que o criador da internet, Tim Berners-Lee, fundou o *World Wide Web Consortium* (W3C), uma organização sem fins lucrativos e livre de royalties que visava criar padrões e protocolos para melhorar a qualidade da web. Desde então o consórcio permanece como a principal organização internacional de padrões de organização da web. A organização já emitiu uma série de padrões, protocolos e recomendações que foram universalmente adotados em sistemas web ao redor do mundo, como HTML, XML, RDF, e OWL[6]. Dentre os grupos de trabalho do W3C, o *Provenance Working Group* foi criado com a intenção de criar padrões para a representação e comunicação de informações de proveniência em sistemas web. Em 2013, o grupo, liderado pelos professores Luc Moreau e Paul Groth, emitiu um conjunto de doze documentos que constituem a recomendação W3C-PROV para representação de proveniência. No centro do padrão PROV está o modelo de dados W3C-PROV-DM, que especifica como são expressados os diferentes elementos e relações de proveniência (Figura 2.2, adaptada de [4]).

[6] https://www.w3.org/standards/

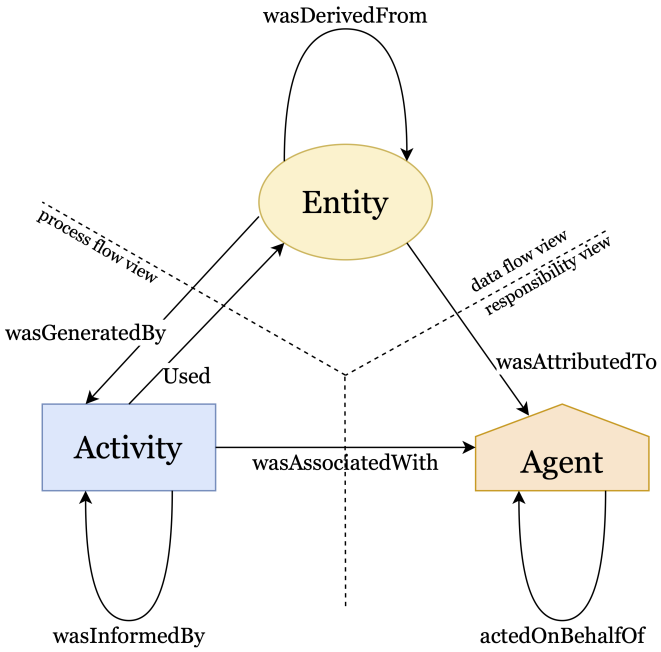


Figura 2.2: Visão esquemática dos elementos e relações do modelo W3C-PROV-DM.

A definição do modelo é mantida intencionalmente simples, com somente três elementos e sete relações básicas, para promover a sua adoção. Temos as seguintes definições para cada elemento PROV [4]:

- **entidades** são literalmente *coisas* digitais, físicas, conceituais ou de outro tipo. Entidades podem ser ser reais ou imaginárias e possuir um conjunto de atributos. A parte do modelo PROV que concerne entidades pode ser definida como a "visão do fluxo de dados" (*data flow view*), que foca no fluxo de informação e na transformação dessas entidades em sistemas. Nesse sentido, entidades são artefatos digitais ou coisas arbitrárias cuja proveniência será descrita. Exemplos: um artigo, uma imagem em JPEG, e um arquivo de sequências biológicas são todos entidades;

- **atividades** são algo que acontece ao longo de um período de tempo e agem sobre ou em conjunto com entidades. Pode incluir o consumo, processamento, transformação, modificação, relocação, utilização, ou geração de tais entidades. Exemplos: criar figuras, executar testes estatísticos, e rodar um programa para alinhamento de sequências são atividades; Podemos definir a parte do modelo relativa as atividades como a "visão do fluxo de processos" (*process flow view*), que refina a visão do fluxo de dados ao incluir as atividades e como se relacionam com entidades. Essa parte do modelo enumera todas as atividades que ocorreram, o seu tempo de início e fim, e seus atributos;

- **agentes** são algo ou alguém que carregam alguma forma de responsabilidade pelo acontecimento de alguma atividade, pela existência de uma entidade, ou pela atividade de outro agente. Logo, a parte do modelo que trata de assinalar a responsabilidade de agentes sobre outros elementos de proveniência é definida como a "visão de responsabilidade" (*responsibility view*). Exemplos de agentes podem ser pessoas, instituições, ou computadores que são responsáveis por alguma entidade, atividade ou por outro agente.

As interações entre esses três diferentes aspectos do modelo PROV são sumarizadas pelas relações entre os elementos de proveniência (Tabela 2.1). Essas relações podem ser classificadas como **não-qualificadas** ou **qualificadas**. A primeira dessas é usada para expressar um relacionamento direto entre os dois elementos envolvidos. As relações qualificadas, por sua vez, são usadas para refinar relacionamentos, sendo possível criar um elemento próprio para o relacionamento que permite a declaração de atributos arbitrários.

Tabela 2.1: Relações de proveniência qualificadas e não-qualificadas do modelo W3C-PROV.

Nome da relação não-qualificada	**Nome da relação qualificada**	**Elementos envolvidos**
actedOnBehalfOf	Delegation	Agent ⇒ Agent
used	Usage	Activity ⇒ Entity
wasAssociatedWith	Association	Activity ⇒ Agent
wasAttributedTo	Attribution	Entity ⇒ Agent
wasGeneratedBy	Generation	Entity ⇒ Activity
wasInformedBy	Communication	Activity ⇒ Activity
wasDerivedFrom	Derivation	Entity ⇒ Entity

Através dessa combinação de elementos e relações de proveniência, é possível expressar toda uma gama de informações a respeito da proveniência de um determinado projeto. Para entender melhor como o formato W3C-PROV funciona na prática, vamos supor o seguinte cenário: uma médica do Hospital Universitário da UFRJ está investigando casos do que ela supõe ser uma nova doença infecciosa. No entanto, a única forma de conseguir um diagnóstico confiável é através de dados de sequenciamento de DNA dos pacientes. Para isso, ela entra em contato com um colega da Fiocruz, responsável por fazer a extração e sequenciamento de DNA, e uma colega no Laboratório Nacional de Computação Científica (LNCC) , que desenvolveu um *workflow* de bioinformática para analisar tais dados de sequenciamento, e os três prontamente iniciam um projeto de pesquisa para confirmar se os novos casos que estão surgindo no hospital são a mesma doença. No mesmo dia que a médica faz as ligações para coordenar esse esforço, dois novos casos aparecem no hospital. Imediatamente são coletadas as amostras de cada paciente, que são transportadas para a Fiocruz, onde o sequen-

ciamento é feito no dia seguinte. Os dados de sequenciamento são transmitidos para o LNCC, a bioinformata roda o *workflow* bioinformático, gera um relatório de resultados que é enviado para a médica. Os resultados da análise bioinformática confirmam: as infecções são causadas pela mesma cepa de bactéria, que apresenta resistência a antibióticos. Com isso, a doutora emite um laudo médico que irá direcionar o tratamento dos pacientes.

Nesse cenário, podemos identificar vários elementos de proveniência; a começar pelas **entidades**: as amostras dos pacientes, os dados de sequenciamento, o relatório de resultados, e o laudo médico. Todas essas são *coisas*, concretas ou digitais, que contém informação relevante. Em seguida, podemos identificar as **atividades** associadas a essas coisas. O processo em questão inicia a partir das amostras coletadas dos pacientes; todas as outras entidades são derivadas destas ou foram geradas por alguma determinada atividade. Por último, podemos identificar os **agentes**, que são pessoas ou instituições envolvidas. Temos três profissionais, de três instituições diferentes, onde cada um é responsável por uma etapa do processo, além dos pacientes, cada um com responsabilidades ou atribuições distintas no processo (Figura 2.3).

Apesar de simples, tal exemplo, de um diagnóstico clínico obtido através do sequenciamento de DNA, faz uso dos três elementos e das sete relações de proveniência, demonstrando como o modelo W3C-PROV pode ser generalizado para as mais diversas situações. Nesse exemplo, o *workflow* bioinformático (assim como o sequenciamento de DNA, e, de certa forma, o diagnóstico) é reduzido a uma mera atividade no contexto geral do diagnóstico. No entanto, como foi estabelecido nas Seções anteriores, um *workflow* como esse pode ser bem complexo, como numerosos passos de manipulação e algoritmos especializados para a análise de sequências. A biblioteca BioProv, apresentada no capítulo seguinte, constrói documentos de proveniência que identificam as entidades, agentes e atividades específicas do WFB, como que se dando um zoom na atividade "*Workflow* bioinformático" apresentada na Figura 2.3. Essa capacidade de poder ser utilizado para detalhar processos em diferentes escalas (por exemplo, "diagnóstico clínico" em comparação com "*workflow* bioinformático") é um aspecto importante do modelo W3C-PROV, pois prova sua versatilidade em vários domínios de aplicação.

No exemplo da Figura 2.3, toda a informação de proveniência está contida em um único documento de proveniência. O modelo W3C-PROV permite a criação de *bundles*, que são subdocumentos que compartimentalizam informações a respeito de um determinado tópico do documento principal. Para esse exemplo específico, poderia-se criar um *bundle* para a atividade "sequenciamento de DNA", um para a atividade "*workflow* bioinformático", que são duas atividades

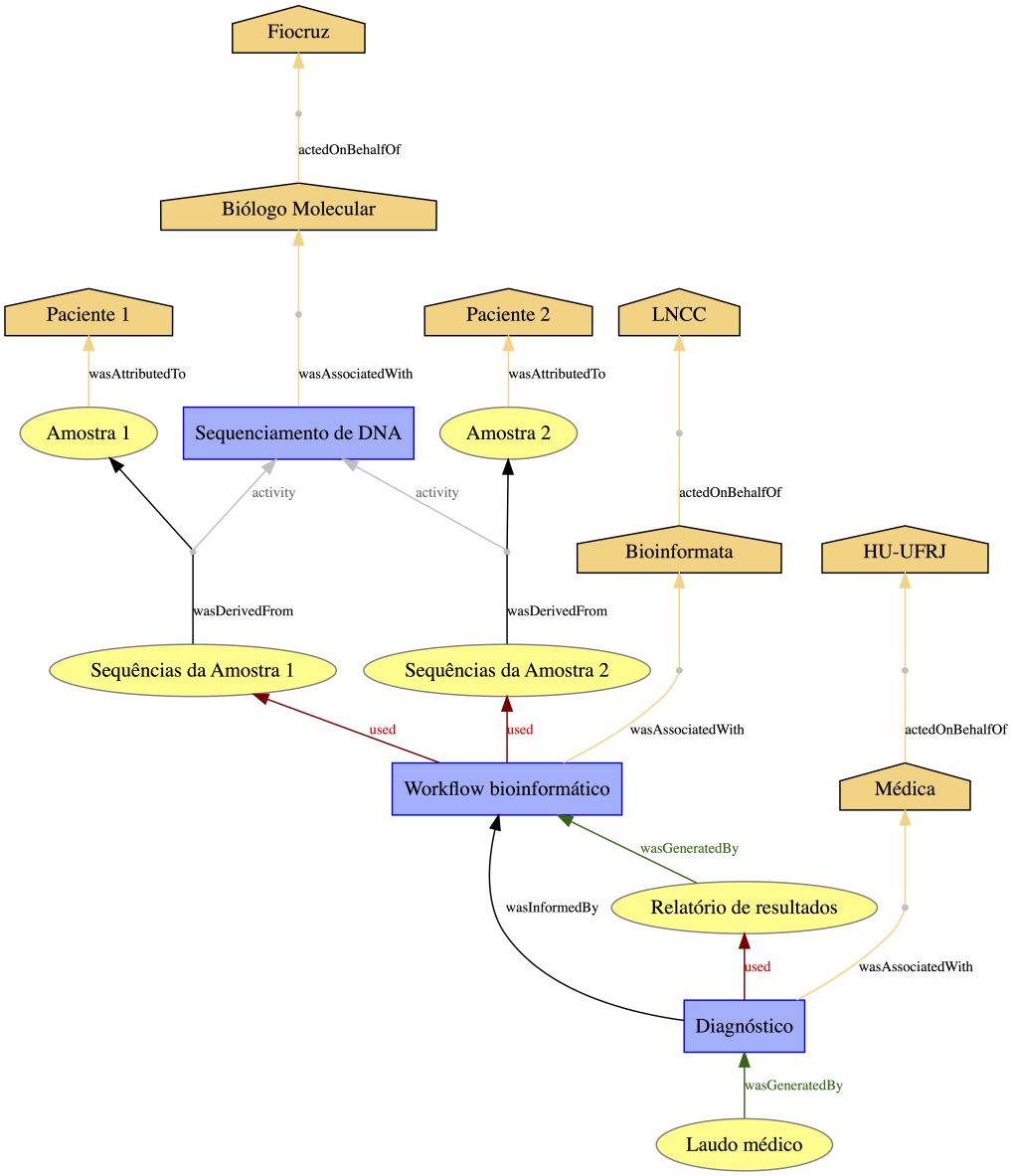


Figura 2.3: Grafo de proveniência no formato W3C-PROV, representando um diagnóstico clínico através do sequenciamento de DNA. As elipses amarelas indicam entidades (coisas), os retângulos azuis indicam atividades (processos ou análises), e os pentágonos laranjas indicam agentes (pessoas e instituições). As setas indicam os diferentes tipos de relações entre estes elementos.

abrangentes e que podem ser descritas com alto nível de detalhamento. Outros elementos que poderiam ser descritos em *bundles* poderiam ser as entidades dos pacientes, que poderiam agregar outros dados pertinentes, como metadados clínicos ou de entrada e saída no hospital, ou até dados confidenciais que poderiam estar protegidos por criptografia, isolados dentro do *bundle*. *Bundles* são, por si, entidades, portanto expressam "proveniência de proveniência" [4].

As decisões de como modelar a proveniência de um processo (que entidades, atividades, e agentes serão registrados, como serão os relacionamentos, e outras especificidades) e de como organizar o documento de proveniência correspondente são fundamentais para tirar o máximo proveito das funcionalidades do formato W3C-PROV. Cabe aos usuários do sistema decidir qual é a melhor forma de descrever o processo, quais são as informações mais importantes a serem coletadas, e qual é o nível de detalhamento necessário, proporcionando imensas possibilidades e dificultando a determinação de uma abordagem ótima. A biblioteca BioProv, apresentada no próximo capítulo, implementa uma de inúmeras modelagens possíveis para descrever um WFB no formato W3C-PROV (essa modelagem é descrita na Seção 3.1.1 e detalhada na 2.3.1). Dificilmente usuários vão considerá-la a abordagem perfeita para as suas necessidades, no entanto foi intencionado que a implementação apresentada nessa dissertação tente conciliar as questões ontológicas tanto do domínio da bioinformática quanto da proveniência em si.

2.3.2 Sistemas de gerência de *workflow* científico

Sistemas de gerência de *workflow* científico (SGWFs) são uma classe de *software* utilizada para gerenciar, representar e executar *workflows* científico, e que, nas últimas duas décadas, se tornaram fundamentais para a pesquisa computacional em múltiplas áreas da ciência [17]. Como consequência, o monitoramento e a captura de dados de proveniência se tornaram atividades fortemente associadas ao SGWF empregado [26], pois este representa a aplicação responsável pela execução do *workflow* e, por extensão, pelo processamento dos dados. A decisão de qual SGWF é mais adequado para a modelagem de um determinado *workflow* torna-se, portanto, relevante para indíviduos e organizações, representando uma peça fundamental no *stack* de tecnologias que são implementadas para um projeto [57]. Esses sistemas geralmente expressam informações de proveniência através de logs e outros arquivos como relatório, grafo e traço de execução. No entanto, nenhum SGWF popular suporta a funcionalidade nativa de coleta de proveniência no formato W3C-PROV, tampouco apresenta a capacidade de especificar com qual granularidade esses dados devem ser coletados. É necessário realizar isso através de middleware especializado, como já foi demonstrado anteriormente [26, 35, 56, 58].

Com a intenção de caracterizar SGWFs como ferramentas importantes para a tarefa de coleta de proveniência e para o desempenho da pesquisa computacional no geral, serão apresentados três SGWFs que foram concebidos no domínio da bioinformática e se tornaram extremamente populares no campo [59].

Nextflow

Nextflow [60] é um SGWF implementado em Apache Groovy [7], uma linguagem da plataforma Java, e que emprega um modelo de "fluxo de dados", baseado em princípios da programação funcional reativa, no qual cada tarefa do *workflow* é isolada em um contexto próprio de execução. Similarmente aos *pipes* da plataforma UNIX[8], as saídas de uma determinada tarefa são automaticamente canalizadas no próximo processo, tornando o paralelismo uma consequência explícita desse modelo. Outra vantagem é que dispensa a necessidade de computar um grafo acíclico direcionado (DAG) , um requerimento comum em outros SGWFs, mas que pode implicar em altos custos de armazenamento para computações muito grandes. Atualmente, Nextflow é discutivelmente o SGWF mais popular entre usuários de bioinformática [59], juntamente de Snakemake, apesar dos dois apresentarem paradigmas constrastantes. Essa popularidade também é demonstrada pela comunidade em volta da plataforma, que conta com um repositório de *workflows* curados que aderem a padrões estritos de qualidade, o Nf-core [61].

Snakemake

Snakemake [62] é um SGWF baseado no paradigma "make" de programação, que é uma linguagem de domínio declarativa utilizada pelo programa homônimo da plataforma UNIX[9]. Similarmente às "Makefiles" processadas pelo GNU Make para compilar e instalar um determinado pacote de *software*, o Snakemake apresenta "Snakefiles", arquivos escritos em uma linguagem de domínio que são processadas pela aplicação CLI do SGWF, que computa o DAG do *workflow* e determina a ordem de execução das operações e se podem ser paralelizadas ou não. Snakemake foi o primeiro SGWF a alavancar a sintaxe de caracteres de *wildcard*, comum em scripts Shell, para inferir automaticamente o nome de amostras ou arquivos a serem processados. Apesar de estabelecer uma linguagem de domínio própria para as Snakefiles, Snakemake é implementado totalmente em Python, o que usuários percebem como uma vantagem [59], presumivelmente dada a popularidade da linguagem entre usuários de bioinformática [63].

Toil

Toil [64] é um SGWF baseado na linguagem CWL (*Common Workflow Language* [65]), que é utilizada em muitos sistemas, mais notavelmente em sua implementação de referência, a ferramenta cwltool[10]. Toil é especializado para a análise de

[7] https://groovy-lang.org/
[8] https://en.wikipedia.org/wiki/Pipeline_(Unix)
[9] https://www.gnu.org/software/make/manual/html_node/Overview.html
[10] https://github.com/common-workflow-language/cwltool

grandes conjuntos de dados biomédicos, e também oferece suporte parcial para WDL (Workflow Domain Language), outra linguagem padrão para *workflows* científicos. Toil é implementado totalmente em Python mas consome scripts declarados em CWL, e, similarmente ao Nextflow, é baseada em um paradigma de programação funcional. Em contraste com Snakemake, ambos estes sistemas também ofereceram suporte nativo a plataformas de nuvem desde seus lançamentos (atualmente essa funcionalidade já existe também para o Snakemake).

2.3.3 Bibliotecas relacionadas

Como já foi discutido, existem bibliotecas que servem como middleware especializado para a captura de proveniência. Essas aplicações geralmente agem na interface entre o ambiente computacional, seja um desktop comum, um serviço distribuído em nuvem ou um cluster de alto desempenho, e o código do workflow. Porém, uma dificuldade comumente encontrada é a dependência entre o SGWF e o modelo de captura de proveniência; uma solução para isso é adotar padrões interoperáveis (como o W3C-PROV) e implementar soluções agnósticas de SGWF [26]. Quando se trata de bioinformática, outra preocupação é a capacidade de extração de dados de domínio a partir de arquivos de dados biológicos. A seguir são apresentadas bibliotecas que tem utilidades importantes para lidar com ambas essas questões: a captura de dados de proveniência e a extração de dados específicos de domínio.

DfAnalyzer

DfAnalyzer (DfA) [58] é um *software* para captura, monitoramento e consulta de dados de proveniência em aplicações intensivas de dados. DfA funciona ao implementar um banco de dados MonetDB e extratores de dados, que capturam os dados brutos e de proveniência e os inserem no banco de dados através de requests HTTP. Tais operações podem ser feitas através de bibliotecas com suporte para as linguagens C++ e Python [66]. O pacote também oferece uma interface gráfica de consultas que pode ser executada em um navegador web. O banco de dados segue um esquema compatível com W3C-PROV para modelar o fluxo de dados da aplicação, e as consultas podem ser feitas durante o tempo de execução de forma online. Apesar de DfA ser uma solução consolidada para a captura de proveniência[11], e que pode ser escalada para ambientes massivamente paralelos, uma de suas limitações é a falta de funcionalidades especializadas para domínios de aplicação, tal como a bioinformática. Isso leva a necessidade de se escrever parsers e outros utilitários para a extração de dados de domínio, o que pode ser

[11] Um exemplo de uso é uma aplicação Spark desenvolvida para *sales forecast* [67].

extremamente custoso tanto para usuários de bioinformática quanto para especialistas em proveniência.

CNNProv

De forma a especializar a DfA para aplicações de aprendizado profundo, foi desenvolvida a biblioteca CNNProv [68], que é voltada para a captura de hiperparâmetros em redes neurais convolucionais. Tais modelos computacionais são usados para resolver problemas diversos, como classificação de imagens e sons, e processamento de linguagem natural [69]. Sua arquitetura é constituída por múltiplas camadas e o treinamento é feito pelo algoritmo de propagação reversa, que impõe uma função de perda cuja otimização influencia a performance da rede. A consequência disso é um modelo razoavelmente complexo, que pode apresentar dúzias de hiperparâmetros, que precisam ser constantemente avaliados para melhorar o desempenho do modelo [70]. A CNNProv apresenta funções que controlam a biblioteca DfAnalyzer para a captura e armazenamento de hiperparâmetros característicos de redes neurais convolucionais. Isso permite que esses dados sejam consultados e usados no treinamento da rede. Um dos componentes da CNNProv é o KerasProv; Keras é uma biblioteca Python para aprendizado profundo[12] de código aberto. O KerasProv é uma extensão do código fonte do Keras que implementa classes para a captura de dados de proveniência dos modelos através da DfAnalyzer. A CNNProv se destaca como um exemplo bem-sucedido de integração de uma biblioteca existente de captura de proveniência com um domínio específico de aplicação.

PROV

PROV [71] é um pacote Python para a criação de documentos de proveniência no formato W3C-PROV. Foi desenvolvido pelo grupo de pesquisa associado ao Provenance Working Group da W3C, da Universidade de Southampton. Esse pacote implementa classes que modelam elementos e relações de proveniência, e serializações em diversos formatos (XML, JSON, PROV-N) para exportação dos documentos. Também permite a exportação de formatos gráficos, como o formato de grafos DOT[13], PDF e PNG. No entanto, assim como DfAnalyzer, é uma biblioteca genérica, sem utilidades especializadas.

[12] https://keras.io/
[13] https://en.wikipedia.org/wiki/DOT_(graph_description_language)

BioPython

BioPython [72] é um conjunto de ferramentas para biologia computacional escrito em Python. A biblioteca apresenta uma série de módulos e componentes para manipular e realizar operações em uma diversidade de dados biológicos, como sequências, alinhamentos, arvores filogenéticas, estruturas proteicas, e modelos matemáticos para genética populacional. Isso permite a leitura e escrita de arquivos em formatos comuns na bioinformática, e a extração semiautomática de dados de domínio. Especialmente importante para essa dissertação são os módulos para processamento de sequências e alinhamentos, que são apresentados no Capítulo 3. BioPython é uma de muitas bibliotecas que visam reunir ferramentas comuns para bioinformática, e que estão disponíveis em diversas outras linguagens, como BioJava [73], BioJulia [74], Rust Bio [75], entre outras.

2.4 Sumário

A bioinformática se desenvolveu como uma ciência impulsionada por dados. Quando o processamento, e não mais a geração, desses dados, se tornou a limitação para a descoberta de conhecimento, novos problemas surgiram nessa disciplina. Uma das principais preocupações para análises computacionais é permitir a reprodutibilidade dos experimentos, e um requerimento chave para isso é capturar os dados de proveniência do experimento. *Workflows* de bioinformática apresentam várias características particulares, como heterogeneidade dos dados e algoritmos especializados, que podem dificultar a captura de proveniência. A criação de sistemas de gerência de *workflows* científicos voltados para a comunidade de bioinformática foi fundamental para aprimorar o monitoramento de proveniência nesse domínio de aplicação. No entanto, tais sistemas não apresentam um padrão interoperável para representação de proveniência, sendo limitados a logs e formatos próprios de relatório, grafo e traço de execução. Alguns pacotes realizam a tarefa de capturar proveniência no formato W3C-PROV, mas nenhum é especializado em aplicações de bioinformática. O próximo capítulo apresenta a biblioteca BioProv, cujo propósito é a captura e representação de dados de proveniência para *workflows* de bioinformática e a geração de documentos compatíveis com o formato W3C-PROV. Ao integrar as funcionalidades das bibliotecas PROV e BioPython, BioProv é o protótipo de uma biblioteca que pretende fornecer uma modelagem de WFBs que seja intuitiva, amigável para o usuário e compatível com um formato interoperável e consolidado.

Capítulo 3

A biblioteca BioProv

O presente capítulo descreve o principal desenvolvimento dessa dissertação: a biblioteca BioProv. Essa biblioteca representa um protótipo, ou prova-de-conceito, de um *software* que é capaz de facilitar ou automatizar a captura de proveniência, e é especializado em *workflows* de bioinformática. Através da modelagem dos componentes comuns a WFBs em elementos de proveniência, BioProv abstrai do usuário a tarefa de captura de proveniência, lidando internamente com as relações de proveniência e criando automaticamente documentos compatíveis com W3C-PROV. BioProv também facilita o monitoramento e reprodutibilidade dos *workflows*, ao implementar captura de proveniência em tempo de execução e extração automática das informações do sistema. Permite que o usuário carregue um projeto armazenado com o sistema de gerência de banco de dados (SGBD) interno da aplicação[1], execute tarefas e atualize o projeto automaticamente. A biblioteca pode ser usada de três formas: i) de forma interativa; ii) executando scripts ou iii) usando a aplicação de linha de comando (CLI) que vem junto com a ferramenta. Para o uso de forma interativa, é recomendado o uso de Jupyter Notebooks [77], e para scripts ou linha de comando, pode ser feita a execução diretamente na Shell do sistema, com o apoio de interfaces como o programa `screen`[2].

3.1 Visão geral

BioProv é uma biblioteca Python que facilita a criação de documentos de proveniência no formato W3C-PROV, é especializada para *workflows* de bioinformática, é baseada em formatos interoperáveis, como JSON e CSV, e é construída em cima

[1] Implementado com TinyDB [76].
[2] https://www.gnu.org/software/screen/manual/screen.html

de outras bibliotecas abertas, como BioPython [72], Pandas [78] e PROV [71].

BioProv é uma biblioteca **orientada a objetos**. É baseada no paradigma de programação homônimo [79], no qual os objetos, ou "classes", contém tanto dados, definidos como atributos, e código, que são comportamentos próprios definidos como procedimentos (chamados de "métodos"); Os objetos construídos por BioProv possuem diversas propriedades que visam suportar as especificidades de WFBs, como uma variedade de parsers para formatos de arquivo comuns na bioinformática (provenientes da biblioteca BioPython), métodos para importar e exportar dados em diferentes formatos, e *wrappers*[1] para programas comumente usados em bioinformática. Isso permite que dados específicos de domínio sejam extraídos de forma rápida e eficiente e sejam modelados na recomendação W3C-PROV.

Outra característica da biblioteca BioProv é ser **baseada em projetos**. Um projeto é uma determinada coleção de amostras e/ou arquivos relacionados e que contém a informação de proveniência referente a essas amostras e arquivos, como programas que foram executados, parâmetros desses programas, tempo de execução, arquivos que foram criados, e metadados desses arquivos. Uma vez que um projeto é criado, é simples realizar consultas ou repetir execuções com diferentes parâmetros, como no caso de uma varredura de parâmetros. Os projetos são implementados pela classe `Project`, que será apresentada posteriormente, e são identificados pelo atributo `Project.tag`, que é o identificador único do projeto no banco de dados interno da biblioteca. Os projetos BioProv são serializáveis no formato JSON, o que permite a interoperabilidade como uma variedade de sistemas.

BioProv foi criada tendo em mente principalmente *workflows* de genômica, e mais especificamente, a análise de dados de sequenciamento de alto desempenho, e portanto apresenta limitações com outros tipos de workflow. Nesse tipo de análise, é comum que uma ou mais bibliotecas de sequenciamento sejam construídas a partir de amostras biológicas distintas. Isso ocasiona na criação de múltiplos arquivos que se referem a uma determinada amostra biológica. Uma consequência disso é a necessidade de rodar os mesmos passos de processamento em cada amostra, em *batch mode*[10]. Também é comum a necessidade de ajustar os parâmetros do *workflow* conforme atributos da própria amostra ou um arquivo relacionado a mesma. A biblioteca oferece funcionalidades que permitem realizar ambas essas atividades, e ao mesmo tempo coleta de dados de proveniência que podem ser consultados durante o tempo de execução do workflow.

BioProv pode ser executada em diversos ambientes computacionais. A insta-

lação pode ser feita facilmente através dos gerenciadores de pacote conda e pip, e portanto pode ser configurada em um ambiente de nuvem, e também possui integração com a plataforma web ProvStore (discutido na Subseção 3.2.3). Para execução em clusters distribuídos, o banco de dados interno pode ser instalado em uma partição compartilhada entre diferentes nós, cada um executando uma instância da biblioteca. O uso de múltiplos processadores é possível quando o programa executado por BioProv tem suporte a essa funcionalidade. O paralelismo nativo para tarefas que seriam outrora executadas serialmente pelo mesmo processador é uma funcionalidade desejada para a biblioteca, e que deve ser implementada no futuro.

É importante destacar que **BioProv não é um sistema de gerência de *workflows* científicos.** BioProv é uma biblioteca para **captura de dados e geração de documentos de proveniência de *workflows*.** Porém, para empenhar as suas funcionalidades completas, BioProv serve como um *wrapper*[1] para os programas que serão executados. Isso é feito pelas funções da biblioteca[3] através de chamadas no código Python. Diferente de SGWFs que possuem uma linguagem de domínio própria e uma máquina de *workflows* que resolve a maneira mais eficiente de executar as tarefas, a lógica de execução deve ser especificada pelo usuário no código Python. Isso apresenta vantagens e desvantagens. As desvantagens envolvem principalmente a necessidade de o usuário ter de especificar manualmente a ordem de execução das tarefas, porém isso é compensado pelas vantagens de poder manipular a lógica do *workflow* com todas as funcionalidades da linguagem Python, usando laços, operações de leitura e escrita de arquivos[4], e a possibilidade de interação com outras bibliotecas Python. Não existe a necessidade de se aprender uma nova sintaxe, uma vez que se entende a API dos objetos BioProv. Isso também permite a execução em ambientes interativos Jupyter, o que facilita o desenvolvimento e *debugging* de *workflows*, especialmente em ambientes de nuvem [56].

3.1.1 Arquitetura de *software*

Para capturar e representar a proveniência de WFBs, BioProv é instalada no ambiente computacional no qual é executado o workflow. Devido a facilidade de instalação através de gerenciadores de pacote (ver a Seção 3.2), pode ser instalada em qualquer tipo de ambiente computacional, como um cluster de alto desempenho ou serviço distribuído de nuvem. É necessário então instrumentar o *workflow* em um script Python, que importa as amostras e define os programas a serem

[3] Por exemplo, pelo método `Program.run()`, ver a discussão na Subseção 3.2.2.
[4] Um exemplo disso é apresentado no Subseção 4.2.2 e no Snippet 4.1.

executados (o Capítulo 4 apresenta um exemplo), ou selecionar um *workflow* pre-existente disponível pela aplicação CLI da biblioteca. As chamadas feitas para a biblioteca vão capturar a proveniência e armazenar a informação do *workflow* em um objeto BioProv, e as entradas serão inseridas no banco de dados interno. Para execução em ambientes distribuídos, múltiplas instâncias de BioProv podem ser configuradas para compartilhar um mesmo banco de dados. Os documentos de proveniência podem ser gerados durante ou ao final do término do workflow, ou criados posteriormente. A realização das consultas ou a exportação de documentos de proveniência pode ser feita através de um ambiente Jupyter, no qual é possível rapidamente carregar o projeto do banco de dados e realizar as operações desejadas (Figura 3.1).

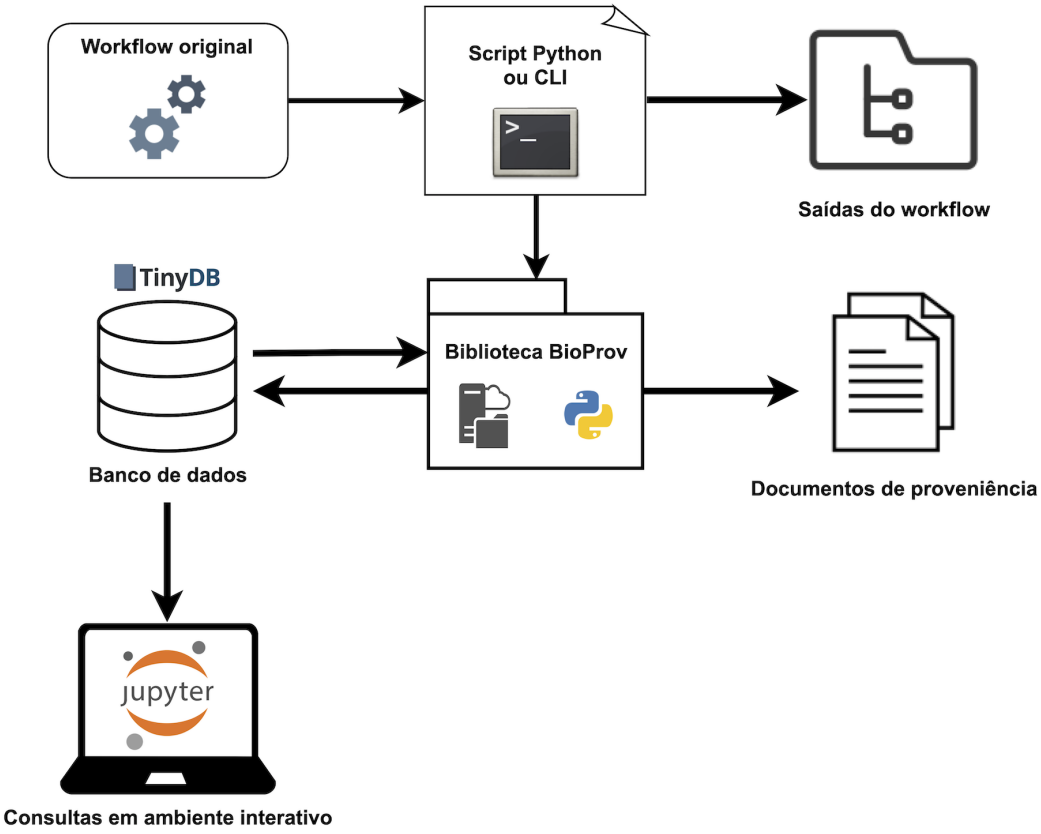


Figura 3.1: Arquitetura de *software* da BioProv.

Para modelar a proveniência de um workflow, BioProv cria um projeto que vai conter amostras, arquivos e programas. Amostras são entidades biológicas que também podem ter arquivos e programas associados a ela. Em análises bioinformáticas, é comum que múltiplos arquivos que se referem a uma mesma amostra biológica contenham dados em formatos diferentes ou coletados por metodologias distintas. Com o objeto `Sample`, é possível agregar arquivos referentes a uma amostra e os programas usados para processar tais arquivos. O objeto `Project`, por sua vez, agrega amostras e também arquivos e programas que se

referem à zero ou mais de uma amostras, *i.e.* não estão associados a uma única amostra. Esses dois objetos estão no centro da arquitetura da biblioteca. A Figura 3.2 ilustra as relações de herança e agregação entre os objetos implementados pela biblioteca. Projetos são agregadores de amostras, arquivos e programas. Amostras, por sua vez, agregam somente arquivos e programas. Projetos podem ser escritos no banco de dados da aplicação (`db.json` na figura), e podem ser exportados como um documento de proveniência.

Funcionalmente, é possível dividir BioProv em 6 componentes funcionais, destacados na Figura 3.2: **Core**, **Arquivos**, **Programas**, **Proveniência**, **Configuração e armazenamento** e **Auxiliares**. O primeiro componente contém os objetos `Project` e `Sample` mencionados acima. Os componentes de Arquivos e de Programas contém classes especializadas para modelar arquivos e programas.

No componente de Arquivos, a classe `SeqFile` tem a capacidade de agregar dados de domínio, especificamente de sequências ou alinhamento de sequências biológicas, através das classes `SeqRecord` e `Alignment`, da biblioteca BioPython. O componente de Programas contém algumas classes auxiliares para criação de programas, o que permite a configuração de presets [5]. O componente de Proveniência é usado para criar documentos de proveniência a partir de um objeto `Project`. A classe `BioProvDocument` é usada para esse fim, ao ser composta por um objeto `PROVDocument`, da biblioteca externa PROV. Esse documento será composto por múltiplos *bundles*, subdocumentos que agregam elementos de proveniência (para uma discussão mais detalhada de *bundles*, referir à Subseção 2.3.1). É importante notar que as funcionalidades de leitura e escrita no formato JSON e de atualização em tempo real estão restritas ao objeto `Project`. Para criação de documentos de proveniência, é necessário exportar o documento ao final da execução. Depois de exportado, o documento pode ser armazenado na web, através da API do serviço ProvStore (Subseção 3.2.3). A Figura 3.3 representa o ciclo de vida típico de um *workflow* modelado com BioProv. Este começa com os arquivos de dados iniciais e os programas que devem ser executados. Isso é modelado em um projeto BioProv, com o qual se pode interagir em um ambiente Python. Esse projeto pode ser lido e escrito no banco de dados da aplicação ou em arquivos, e é atualizado em tempo de execução. Ao final da execução, é criado um documento de proveniência no formato W3C-PROV, que pode ser armazenado na web ou em disco.

O componente de Configuração e armazenamento contém as configurações de instalação e controla o SGBD da aplicação. A classe `BioProvDB` herda da bi-

[5] A biblioteca vem com vários programas pré-configurados, listados na Tabela 3.5

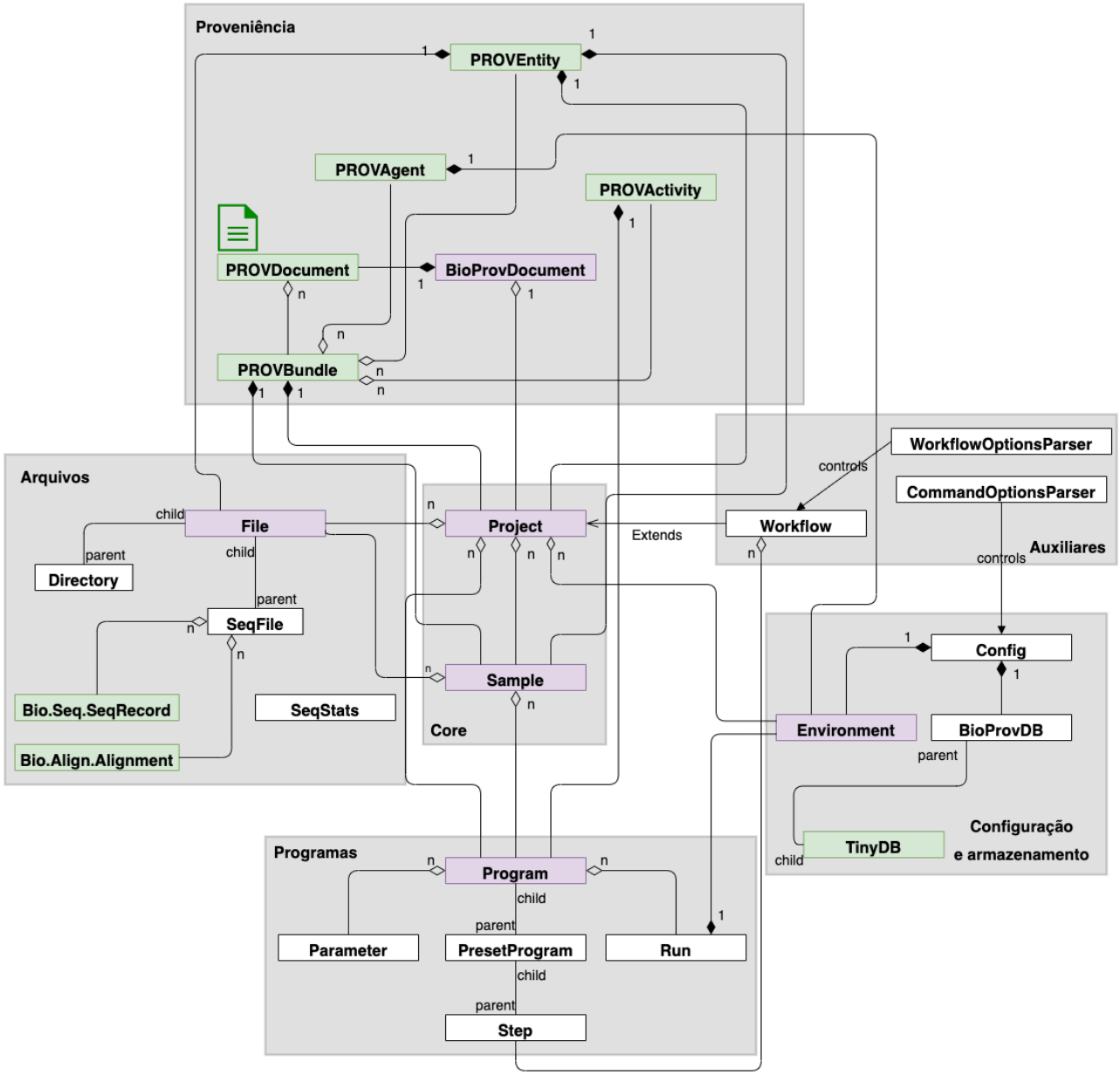


Figura 3.2: Diagrama de classes BioProv, com as relações de herança e agregação entre as diferentes classes. Em roxo: classes principais; em verde: classes de bibliotecas externas; em branco: classes auxiliares. As relações representadas podem ser de agregação (losango com número ao lado), de herança (relação "parent-child"), de extensão ou controle. Os losangos pretos representam relações de cardinalidade obrigatória, enquanto os losangos transparentes representam relações de cardinalidade opcional.

blioteca TinyDB e cria um banco de dados orientado a documentos local. Os projetos BioProv podem então serem armazenados ou carregados desse banco. O componente de Auxiliares é usado para controlar a aplicação CLI da biblioteca, e também possui a classe `Workflow`, que extende a classe `Project` para criar *workflows* pré-configurados.

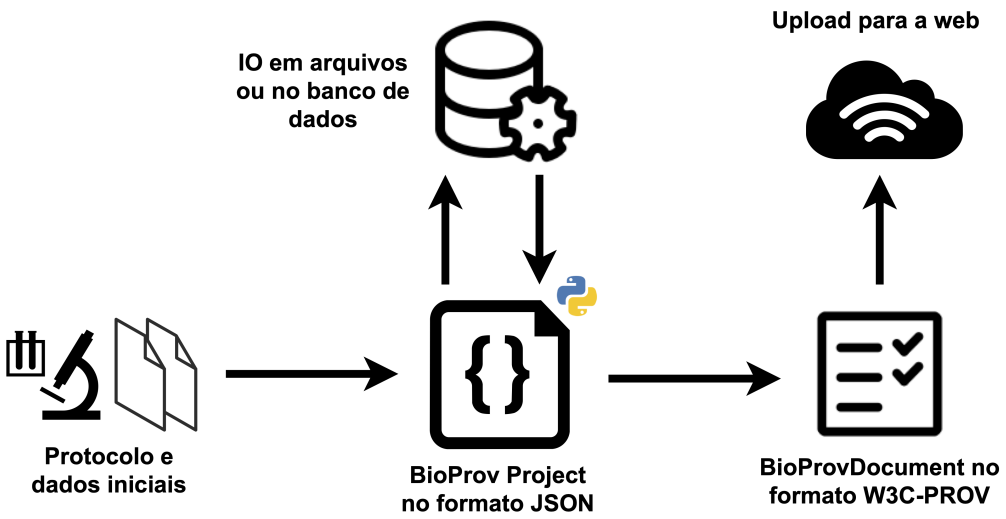


Figura 3.3: Ciclo de vida de um *workflow* modelado com BioProv.

A biblioteca é dividida em cinco subpacotes (Tabela 3.1), mas as funcionalidades principais moram no subpacote `src`, que contém a lógica das classes usadas para extrair e armazenar a proveniência de WFBs. Os outros subpacotes são auxiliares, contendo dados de exemplo, presets e testes.

Tabela 3.1: Subpacotes da biblioteca BioProv.

Nome	Propósito
`data`	Subpacote com dados de exemplo
`programs`	Presets de programas comumente utilizados
`src`	Pacote principal contendo o código fonte das classes
`tests`	Testes unitários e de integração
`workflows`	Presets de *workflows* comumente utilizados

São implementadas um total de dezoito classes (Tabela 3.2), divididas em dois tipos: seis classes **principais**, que modelam os elementos de um WFB de forma correspondente a um elemento de proveniência, e onze classes **auxiliares**, que gerenciam funcionalidades internas da biblioteca ou herdam das classes principais, sendo especializações das mesmas. Uma descrição detalhada de cada classe é apresentada na Subseção 3.2.2. Apesar da diversidade de classes e objetos implementados pela biblioteca, pode-se resumir seu uso geral através de quatro classes principais: **Projects**, **Samples**, **Files**, e **Programs**. **Project** é um objeto *top-level* que vai encapsular toda a informação relacionada a um projeto. Um projeto geralmente contém amostras biológicas, que são descritas pelo objeto **Sample**. Uma amostra pode conter arquivos (**Files**) ou programas (**Programs**) associados à ela. Se um arquivo ou programa em particular está associado a nenhuma ou mais de uma amostra, ele pode ser associado diretamente ao projeto.

Essa organização vai gerar uma estrutura hierárquica, que pode então ser

Tabela 3.2: Classes implementadas por BioProv. As classes principais possuem elementos de proveniência PROV correspondentes.

Classe	Tipo de classe	Elemento PROV correspondente
`BioProvDocument`	Principal	PROVDocument
`Environment`	Principal	PROVAgent
`File`	Principal	PROVEntity
`Program`	Principal	PROVActivity
`Project`	Principal	PROVEntity
`Sample`	Principal	PROVEntity
`BioProvDB`	Auxiliar	-
`CommandOptionsParser`	Auxiliar	-
`Config`	Auxiliar	-
`Directory`	Auxiliar	-
`Parameter`	Auxiliar	-
`PresetProgram`	Auxiliar	-
`Run`	Auxiliar	-
`SeqFile`	Auxiliar	-
`SeqStats`	Auxiliar	-
`Step`	Auxiliar	-
`Workflow`	Auxiliar	-
`WorkflowOptionsParser`	Auxiliar	-

representada em um arquivo JSON. A Figura 3.4 demonstra uma adaptação da representação de um `bioprov.Project`.

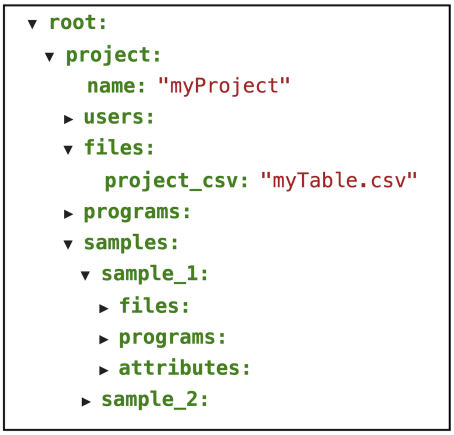


Figura 3.4: O modelo de dados de um projeto BioProv segue uma estrutura hierárquica e serializável em JSON.

Uma vez estruturado dessa forma, o projeto pode ser facilmente atualizado e armazenado. BioProv possui diversas funções que facilitam a leitura e escrita de dados tanto em seu formato nativo (compatível com JSON) quanto em formatos tabulares. Os formatos tabulares, embora muito versáteis para integração com outras ferramentas, omitem algumas informações do projeto, como informações relacionadas aos programas que foram executados. No entanto, eles retém informações relacionadas a atributos e arquivos de amostras.

3.1.2 Exemplo de uso

A seguir apresenta-se um exemplo de uso da biblioteca que poderia ser reproduzido em um ambiente interativo, como um interpretador IPython. Será construído um projeto usando duas amostras que correspondem a dois genomas depositados no banco de dados NCBI RefSeq. No entanto, para fins de simplicidade, vamos nomear as amostras como **sample_1** e **sample_2**. Assumindo uma tabela de dados como a Tabela 3.3, armazenada em um arquivo `myTable.csv`:

Tabela 3.3: Exemplo de tabela usada para importar dados.

sample-id	assembly	report	refseq-accession
sample_1	contigs_1.fasta	report_1.txt	GCF_000007925.1
sample_2	contigs_2.fasta	report_2.txt	GCF_000010065.1

as amostras são identificadas pela primeira coluna, `sample-id`, e tem dois arquivos associados, identificados respectivamente pelas colunas `assembly` e `report`. A coluna `refseq-accession` será interpretada como um atributo da amostra.

Para usar BioProv interativamente ou como scripts, ela deve ser importada como qualquer outra biblioteca Python. Para isso, usa-se a convenção `bp`. Também será importado um preset de programa, referente ao pacote Prodigal [80]. Será criada uma instância do programa associada à `sample_1`, que será executada, e o projeto será exportado:

```
import bioprov as bp
from bioprov.programs import prodigal

project = bp.read_csv("myTable.csv",
                      file_cols="report",
                      sequencefile_cols="assembly",
                      tag="myProject",
                      import_data=True)

with project["sample_1"] as sample:
    sample.add_programs(prodigal(sample))
    sample.run_programs()

project.to_csv()  # exporta em formato tabular

project.to_json()  # exporta em formato BioProv (JSON)
```

Snippet 3.1: Exemplo de uso da biblioteca BioProv para importar um projeto e executar um programa.

A função `read_csv()` é inspirada na biblioteca Pandas, e visa importar rapidamente um projeto BioProv. O objeto construído, `bioprov.Project`, vai conter toda a informação de proveniência relacionada a esse projeto. Isso inclui as amostras biológicas presentes e seus respectivos atributos, arquivos e programas associados às amostras, e arquivos e programas associados ao projeto. Outras informações também incluem as variáveis locais do ambiente do usuário. Os parâmetros `file_cols` e `sequencefile_cols` indicam colunas que contém arquivos, no entanto o primeiro indica colunas com arquivos genéricos e o segundo indica colunas que contém arquivos de sequências biológicas. O parâmetro `import_data` indica que os dados de domínio dessas sequências devem ser importados, o que permite a captura automática de metadados, como quantidade e comprimento das sequências. A coluna `refseq-accession` será importada como um atributo das amostras. Como BioProv é executada em um ambiente Python, pode-se utilizar as funcionalidades da linguagem para manipular, automatizar e paralelizar os processos do workflow. Os objetos BioProv podem ser extendidos e customizáveis de acordo com as necessidades do usuário.

3.1.3 Objetivos

O propósito da biblioteca BioProv é **gerar documentos de proveniência para *workflows* de bioinformática.** Para isso, podemos dividir os objetivos em teóricos e práticos. Os objetivos teóricos envolvem principalmente as decisões de como modelar os dados de proveniência depois de sua extração, em um formato que seja interoperável e amigável tanto para máquinas quanto para humanos; sendo assim, os elementos comuns de WFBs devem ser correspondidos com elementos de proveniência na recomendação W3C-PROV. Os objetivos práticos envolvem fornecer uma implementação prática, moderna e de acordo com boas práticas de *software* aberto; que seja sustentável, e que possa ser continuamente aprimorada, assim proporcionando captura de proveniência de uma forma que seja simples e conveniente para o usuário.

3.2 Detalhes da implementação

BioProv é uma biblioteca totalmente em Python, compatível com Python $\geq$ 3.6. O código fonte, incluindo a versão de desenvolvimento, está disponível no repo-

sitório do GitHub, por onde são feitos os lançamentos da biblioteca (versão atual: v0.1.22). Os lançamentos também são feitos no PyPI (Python Package Index). , e pelo canal Bioconda, que disponibiliza pacotes através do gerenciador de pacotes `conda`. Visando cumprir com boas práticas de *software* aberto [5, 25], BioProv segue o estilo Black de formatação, oferece um guia de contribuição, uma página de documentação, e implementa integração contínua com GitHub Actions, um serviço web automatizado para execução de testes, *linting* e criação de lançamentos. A suíte de testes é executada com Pytest e apresenta, na versão v0.1.22, 93% de cobertura.[6]

Como documentação, além da página de documentação, dois tutoriais são oferecidos (referir ao Apêndice B), que podem ser executados localmente em um ambiente Jupyter/IPython, ou remotamente através do serviço Binder.

BioProv conta com algumas dependências (Tabela 3.4) que são especificadas no arquivo `setup.py`, na raíz do repositório. Através do uso de bibliotecas abertas bem conhecidas, como BioPython e Pandas, é possível aproveitar as amplas funcionalidades dessas bibliotecas para suportar uma gama diversa de formatos de arquivo.

Tabela 3.4: Dependências utilizadas por BioProv.

Dependência	Propósito
`biopython`	Parsers para formatos de arquivo de bioinformática
`coolname`	Geração de nomes aleatórios
`coveralls`	Medir cobertura de testes
`dataclasses`	Compatibilidade com Python 3.6
`pandas`	Leitura e processamento de arquivos tabulares
`prov`	Criação de relatórios de proveniência
`prodigal`	Programa de bioinformática utilizado para testes
`provstore-api`	Integração com API ProvStore
`pydot`	Criação de grafos de proveniência
`pytest`	Framework de testes
`pytest-cov`	Plugin para medir cobertura de testes
`tqdm`	Renderização de barras de progresso
`tinydb`	SGBD interno

Uma funcionalidade importante de BioProv é a capacidade do usuário criar os seus próprios **presets**. Presets estão disponíveis através de duas classes: a classe `PresetProgram`, usada para criar presets de programas [7], que são programas pré-configurados com diversos parâmetros *default*, que necessitam de um mínimo de código para funcionar; e a classe `Workflow`, que é usada para criar

[6] Cobertura de código se refere à porcentagem do código fonte que é executada durante a execução da suíte de testes. Para BioProv, essa rotina leva em volta de 40-60 segundos (Figura 3.7).

[7] Disponíveis através do subpacote `bioprov.programs`.

uma sequência de programas a serem executados nos arquivos de um projeto[8]. Na versão v0.1.22, são oferecidos onze presets de programas (Tabela 3.5), que cobrem algumas tarefas bioinformáticas comuns, como busca por alinhamento, alinhamento múltiplo e predição de genes. É importante notar que, com a exceção do Prodigal, esses programas não são distribuídos com BioProv, devendo ser instalados separadamente. A razão por trás disso é poder distribuir BioProv em um arquivo binário menor e com menos dependências.

Tabela 3.5: Presets (programas pré-configurados) disponíveis na versão v0.1.22.

Programa	**Propósito**
BLASTn [81]	Busca de sequências por alinhamento
BLASTp [81]	Busca de sequências por alinhamento
DIAMOND [82]	Busca de sequências por alinhamento
FastTree2 [83]	Árvores probabilísticas de máxima verossimilhança
Kaiju [84]	Anotação taxonômica de metagenomas
`kaiju2table` [84]	Anotação taxonômica de metagenomas
Kallisto [85]	Quantificação de transcritos
MAFFT [86]	Alinhamento múltiplo de sequências
MUSCLE [87]	Alinhamento múltiplo de sequências
Prodigal [80]	Predição de genes procarióticos
Prokka [88]	Anotação de genomas procarióticos

Os presets de *workflows*[9] são ideais para a criação rápida de um WFB no qual existem N amostras e, para os arquivos referentes à cada amostra, serão executados um sequência de k programas [10]. No entanto, é importante notar que BioProv não é um sistema de gerência de *workflows* científicos, e sim uma biblioteca de captura de proveniência, logo não possui funcionalidades para a execução otimizada das tarefas de um workflow; para isso, BioProv deve ser utilizada juntamente de uma ferramenta que exerça a função de SGWF, como Snakemake [62] ou Nextflow [60].

3.2.1 Subpacotes e módulos

A biblioteca é divida em cinco subpacotes (Tabela 3.1), embora todos os componentes fundamentais morem no subpacote `src`. BioProv também possuem dois

[8] É importante fazer a distinção entre os termos "projeto" e "workflow" no uso da biblioteca BioProv. Um projeto se refere a um grupo de amostras e/ou arquivos interrelacionados, enquanto um *workflow* se refere a um conjunto de programas a serem executados nos dados do projeto. Isso é correspondente a visão do fluxo de dados e visão do fluxo de processos do modelo W3C-PROV (Figura 2.2)

[9] Disponíveis através do subpacote `bioprov.workflows` ou através da aplicação CLI.

[10] Essa é uma operação comum em *software* de *workflows*, conhecida como *batch mode*, ou "modo de lote". Nesse caso, as tarefas a serem executadas formam um produto cartesiano igual a $N * k$. Os programas também podem processar arquivos a nível de projeto (que estão associados à zero ou mais de uma amostra).

módulos [11] auxiliares na raíz do pacote: `utils.py` e `bioprov.py`, que contém respectivamente diversas funções auxiliares e a lógica da aplicação CLI da biblioteca, e as credenciais para uso da API ProvStore e o banco de dados interno ((Figura 3.5)).

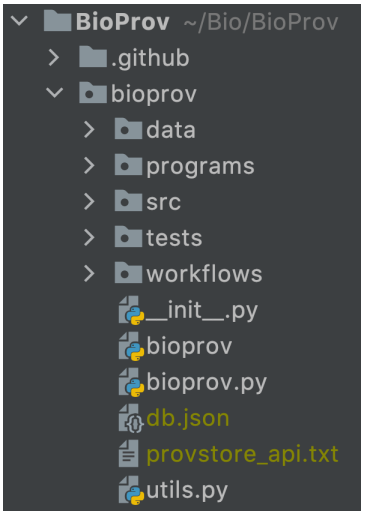


Figura 3.5: Estrutura do diretório que contém o pacote BioProv.

`data`

O subpacote `data` possui dados de exemplo para a instalação e uso da biblioteca. Atualmente, esses dados consistem em:

- cinco genomas bacterianos (diretório `genomes/`);
- uma fração do banco de dados MEGARES [89], formatado como um banco de dados BLAST[12] (diretório `blastdb/`);
- arquivos tabulares que podem ser importados como projetos BioProv (diretório `datasets/`).

O módulo inicializador do subpacote (o arquivo `__init__.py`) contém esses dados formatados de acordo com o caminho da instalação, facilitando a importação dos dados:

```
import bioprov as bp
from bioprov.data import picocyano_dataset,
   synechococcus_genome
```

[11] Na linguagem Python, entende-se como "módulo" um arquivo `.py` que contém objetos Python que podem ser importados e referenciados externamente. Entende-se como "pacote" um diretório com vários módulos e que contém o módulo de inicialização nomeado `__init__.py`.

[12] Ver parágrafo 1.1 do BLAST QuickStart.

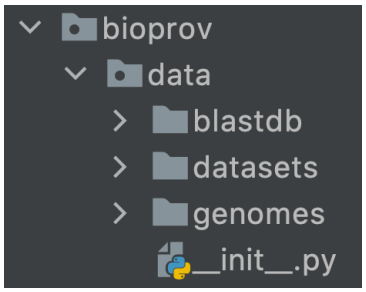


Figura 3.6: Conteúdo do subpacote `data`.

```
# Criando um projeto
proj = bp.read_csv(picocyano_dataset, sequencefile_cols
   ="assembly")

# Criando uma amostra
my_sample = bp.Sample("s_elongatus_pcc_6301",
                      files={"assembly":
   synechococcus_genome})
```

Snippet 3.2: Importando dados presentes no subpacote `data`.

`programs`

O subpacote `programs` é onde ficam os presets de programa (Tabela 3.5), e contém somente dois módulos: `programs.py` e o módulo inicializador. Um exemplo de uso de importação e uso de um preset de programa está no Snippet 3.1.

`src`

O subpacote `src` contém os componentes principais da biblioteca, e é dividido em cinco módulos, além do inicializador:

Tabela 3.6: Módulos do subpacote `src`.

Módulo	Conteúdo
`config.py`	Configurações e opções *default* da biblioteca
`files.py`	Classes que modelam arquivos, diretórios e formatos de arquivos biológicos.
`main.py`	Classes que modelam projetos, amostras, programas e funções de IO.
`prov.py`	Cria documentos de proveniência no formato W3C-PROV.
`workflows.py`	Classes que modelam presets de *workflows*.

O módulo `config.py` contém as configurações da biblioteca, e pode ser usado para ajustar opções como número padrão de processadores a serem usa-

dos, diretórios dos arquivos de dados, credenciais para a API ProvStore (essa funcionalidade é descrita na Subseção 3.2.3), e caminho do banco de dados TinyDB.

No módulo `files.py`, são encontradas as classes que modelam arquivos e diretórios. Esse módulo importa as bibliotecas SeqIO e AlignIO, que são usados para fazer a leitura de formatos de arquivos biológicos. Também possui a `dataclass` [13] `SeqStats`, [14], inspirada no programa `seqstats`, usado para calcular métricas de sequências biológicas, como número total de sequências e pares de base. Outras métricas calculadas pela classe são o N50[15] e conteúdo GC[16].

O módulo `main.py` contém a maior parte dos componentes principais da biblioteca. Isso inclui as classes que modelam projetos, amostras, programas, parâmetros de programas, *workflows*, e funções de entrada e saída de dados. Tais funções (Tabela 3.7) empenham tarefas como escrita e leitura de arquivos tabulares e arquivos JSON, e também interações com o banco de dados interno da biblioteca.

Tabela 3.7: Funções para entrada e saída de dados presentes no módulo `src.main`.

Nome	O que faz
`from_df`	Importa um dataframe Pandas
`from_json`	Lê arquivo JSON
`load_project`	Carrega projeto do banco de dados
`read_csv`	Lê arquivos tabulares
`to_json`	Converte objeto em string JSON
`write_json`	Escreve arquivo JSON

Para uma explicação detalhada dos componentes do módulo `main.py`, recomenda-se ler a Subseção 3.2.2, ou a página de documentação.

O módulo `prov.py` contém a classe `BioProvDocument`, que é usada para construir documentos de proveniência. Esse módulo utiliza extensivamente a biblioteca PROV para criação dos elementos de proveniência. Uma descrição detalhada é encontrada na Subseção 3.2.3 ou na descrição da classe (Subseção 3.2.2).

O módulo `workflow.py` contém as classes `Workflow` e `Step`, que são usados para criar presets de workflow, que por sua vez moram no subpacote *`workflows`*. A primeira classe vai conter a lógica do workflow, como sequência de programas a serem executados, criação do parser da linha de comando, e opções de exporta-

[13] `dataclasses` são classes disponíveis no Python $\geq$ 3.7 que são usadas ara abstrair a criação de classes que contém, exclusivamente, dados expressos como atributos da classe.

[14] `https://github.com/clwgg/seqstats`

[15] N50 é uma métrica de contiguidade de genomas. Uma explicação da medida pode ser encontrada aqui.

[16] Conteúdo GC é a porcentagem de nucleotídeos G e C em relação ao total de nucleotídeos na amostra.

ção. A segunda é uma especialização da classe `Program` adaptada para o uso em presets de *workflows*.

`tests`

O subpacote `tests` contém os testes unitários e de integração da biblioteca. Os testes são divididos em oito módulos e a execução é feita com Pytest (Figura 3.7). A cobertura dos testes[6] é medida com o pacote coverage, e é de 93% na versão atual (Figura 3.8).

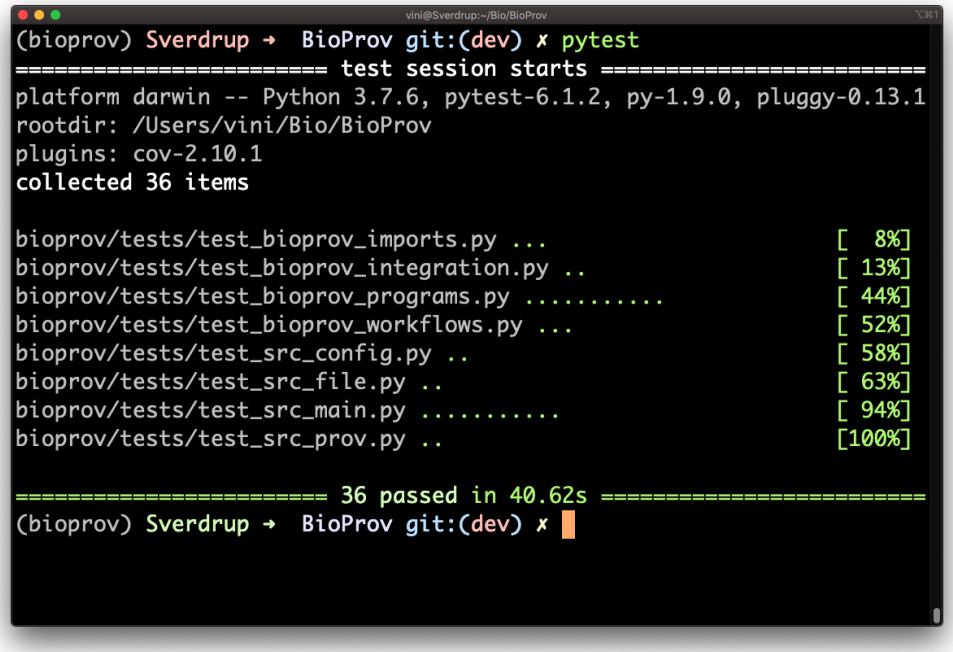


Figura 3.7: Execução da suíte de testes.

`workflows`

O subpacote `workflows` é similar ao `programs`, porém contém presets de *workflows* no lugar de programas. Esse subpacote possui um único módulo `workflows.py` que contém os presets e a classe `WorkflowOptionsParser`, que gerencia o uso de *workflows* na aplicação CLI.

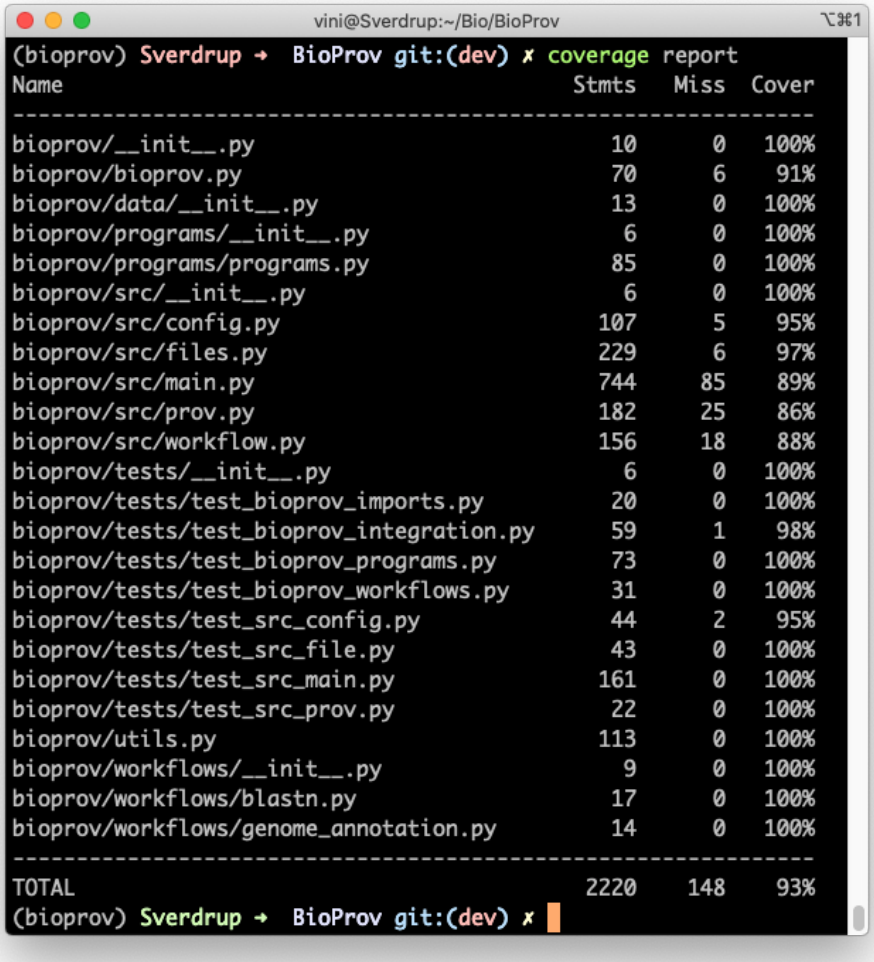


Figura 3.8: Cobertura de código da suíte de testes.

3.2.2 Classes

Na presente subseção é apresentada uma descrição de cada classe presente na biblioteca. Um sumário das classes é apresentado na Tabela 3.2. As seguintes descrições serão reproduzidas na página de documentação da biblioteca, e futuramente essa página que será utilizada como documentação oficial. As classes chamadas **principais** são classes que correspondem a elementos de proveniência no formato W3C-PROV. Essa correspondência é realizada pelos métodos internos da classe `BioProvDocument`, que constrói o documento de proveniência. As classes **auxiliares** são classes que realizam funções internas da biblioteca BioProv, ou classes que especializam ou compõem classes principais para realizar funções adicionais (no caso de `PresetProgram`, `Run`, `Step`, `SeqFile`, e `Workflow`).

BioProvDB

Tipo de classe: auxiliar **Módulo:** `src.config`

Essa classe herda do objeto `tinydb.TinyDB`, que implementa um banco de dados orientado a documentos. O arquivo JSON no qual o banco é armazenado é criado no diretório da instalação de BioProv, e um projeto pode ser chamado através da sua `tag`, que é o seu identificador no banco TinyDB. Isso é feito com a função `load_project()`, do módulo `src.main`.

```
class BioProvDB(TinyDB):
    """
    Inherits from tinydb.TinyDB.

    Class to hold database configuration and methods.
    """

    def __init__(self, path):
        super().__init__(path)
        self.db_path = path
    # [...]

# Carregando um projeto
import bioprov as bp
project = bp.load_project("myProject")
```

Snippet 3.3: Trecho de código fonte da classe `BioProvDB` e e como carregar um projeto com a função `load_project()`.

BioProvDocument

Tipo de classe: principal **Módulo:** `src.prov`

Uma das classes principais da biblioteca, o objeto gerado por `BioProvDocument` irá conter o documento de proveniência compatível com o formato W3C-PROV. O seu propósito é corresponder os objetos BioProv aos elementos de proveniência da biblioteca PROV. O documento principal (invocado pelo atributo `BioProvDocument.PROVDocument`) é um objeto `PROVDocument` da biblioteca PROV. A classe possui diversos métodos internos que cumprem essa função de correspondência e interagem com o objeto `PROVDocument`. Para uma referência completa dos métodos, consultar a entrada da classe na documentação.

```
class BioProvDocument:
    """
    Class containing base provenance information for a
   Project.
    """

    def __init__(
        self,
        project,
        add_attributes=False,
        add_users=True,
        _add_project_namespaces=True,
        _iter_samples=True,
        _iter_project=True,
    ): ...
```

Snippet 3.4: Docstring e método construtor da classe `BioProvDocument`.

Existem alguns parâmetros no método construtor da classe que permitem refinar a quantidade de informação que será adicionada ao documento de proveniência. Por exemplo, as opções `add_attributes` e `add_user` servem para incluir ou omitir informações relacionadas aos atributos e usuários. Os parâmetros `_add_project_namespaces`, `_iter_samples` e `_iter_project` tem a função de ativar ou desativar os métodos internos da classe que iteram sobre os objetos BioProv e criam as relações de proveniência. Uma descrição mais detalhada dessa classe é apresentada na Subseção 3.2.3.

CommandOptionsParser

Tipo de classe: auxiliar **Módulo:** `bioprov.bioprov`

A classe `CommandOptionsParser` é uma classe auxiliar do módulo `bioprov.bioprov`

que é usada para gerenciar a aplicação CLI da biblioteca. É responsável por invocar os comandos disponíveis na aplicação CLI, como `show_config`, `list`, e `clear_db`.

Config

Tipo de classe: auxiliar **Módulo:** `src.config`

A classe `Config` abriga as configurações padrão da biblioteca. Os valores são definidos quando a classe é importada. Caso o usuário queira mudar os valores padrão, é necessário editar os valores no módulo `src.config`. Esses valores podem ser listados com o comando `bioprov -show_config`.

Directory

Tipo de classe: principal **Módulo:** `src.files`

A classe `Directory` é uma classe principal que é usada para extrair informações referentes a diretórios. Assim como arquivos criados com a classe `File`, diretórios podem ser associados a uma amostra ou ao projeto. É modelada como uma entidade de proveniência com a classe PROVEntity da biblioteca PROV.

Environment

Tipo de classe: principal **Módulo:** `src.config`

A classe `Environment` é uma classe principal usada para armazenar informações do ambiente computacional local. É modelada como um agente de proveniência com a classe PROVAgent da biblioteca PROV. O atributo `Environment.user` é usado para extrair informações do usuário quando essas não são fornecidas manualmente. Qualquer tarefa executada com BioProv será associada a um ambiente computacional, que por sua vez será associado a um usuário.

File

Tipo de classe: principal **Módulo:** `src.files`

A classe `File` é uma classe principal que é usada para extrair informações referentes a arquivos. Arquivos são entidades de proveniência que são associadas a uma amostra ou ao projeto (quando o arquivo se refere a zero ou duas ou mais amostras), portanto, assim como a classe `Directory`, são correspondentes a classe PROVEntity da biblioteca PROV. A classe `File` estabelece vários atributos como `.extension`, `.size` e `.exists` que modificam comportamentos da mesma. O atributo `.sha256` é um *checksum* no padrão SHA256 que pode ser usado para verificar modificações nos arquivos.

Parameter

Tipo de classe: auxiliar **Módulo:** `src.main`

A classe `Parameter` é uma classe auxiliar que é usada para construir programas. É usada para definir os parâmetros que serão incluídos na string que será avaliada pela linha de comando do sistema na hora de executar um programa. A classe possui o atributo `.kind` que pode ser declarado como ‘‘`input`’’ ou ‘‘`output`’’, quando o valor do parâmetro se refere a um arquivo de entrada ou saída, respectivamente[17]. Declarar esse atributo corretamente ajuda a biblioteca a refinar as relações de proveniência presentes no documento de proveniência resultante. Embora as strings de comando de um programa possam ser declaradas arbitrariamente (através do atributo `Program.cmd`), é sempre recomendado construi-las de forma programática usando parâmetros (apesar de ser mais custoso), como no exemplo abaixo:

```
import bioprov as bp

input_dir = bp.Directory("an_existing_directory")
output_file = bp.File("content_of_an_existing_directory.txt"
    )

ls = bp.Program("ls")
input_param = bp.Parameter(str(input_dir), kind="input")
output_param = bp.Parameter(">", str(output_file), kind="
    output")
for param in (input_param, output_param):
        ls.add_parameter(param)

ls.cmd
# '/bin/ls/ an_existing_directory >
    content_of_an_existing_directory.txt'
```

Snippet 3.5: Como construir um programa usando a classe `Parameter`

Repare que no primeiro parâmetro adiciona-se somente o valor do arquivo de entrada e no segundo adicionamos a “chave” do parâmetro, que é o operador *redirect* do sistema UNIX[18]. Sempre que a chave de um parâmetro for especificada, ela irá preceder o valor do parâmetro na string que será avaliada pelo sistema.

[17] Um arquivo de entrada é consumido pelo programa e um arquivo de saída é criado pelo programa.

[18] https://www.gnu.org/software/bash/manual/html_node/Redirections.html

PresetProgram

Tipo de classe: auxiliar **Módulo:** `src.main`

A classe `PresetProgram` é uma classe auxiliar que serve para especializar a classe principal `Program`, e é usada para criar presets de programas no módulo `programs.py`. Os programas criados com essa classe já assumem parâmetros de entrada e saída, e criam os arquivos correspondentes para a(s) amostra(s) ou projeto que estão associados.

```
def muscle(sample, input_tag="input", msf=False, extra_flags
    =None):
     """
     :param Sample sample: Instance of BioProv.Sample.
     :param str input_tag: A tag for the input multi-fasta
    file.
     :param bool msf: Whether or not to have the output in
    msf format.
     :param list extra_flags: A list of extra parameters to
    pass to Muscle.
     :return: Instance of PresetProgram for Muscle.
     :rtype: BioProv.PresetProgram.
     """

     _muscle = PresetProgram(
         name="muscle",
         sample=sample,
         input_files={"-in": input_tag},
         output_files={"-out": ("_muscle_hits", "_muscle_hits
    .afa")},
         extra_flags=extra_flags,
     )

     if msf:
         _muscle.add_parameter(Parameter(key="-msf"))

     return _muscle
```

Snippet 3.6: Código fonte do preset do programa MUSCLE. Contribuição do usuário @jvfe.

Uma vez disponível no subpacote `programs`, um preset pode ser importado para uso pelo usuário, que só precisa fornecer a amostra ou projeto para qual o programa será associado (Snippet 3.2).

Program

Tipo de classe: auxiliar **Módulo:** `src.main`

A classe `Program` é uma classe principal que é usada para extrair informações referentes a programas. Programas são tarefas executadas por BioProv que são modeladas como atividades de proveniência, correspondentes a classe PROVActivity da biblioteca PROV. Para facilitar a criação de programas, é possível usar a classe `PresetProgram` e armazenar o preset no subpacote `programs`. Outra especialização da classe `Program` é `Step`, usada para criar passos presentes em um preset de workflow. Atributos definidos por `Program` incluem nome e versão do programa, localização do binário sistema, e a string que será avaliada para executar o programa (como exemplificado no Snippet 3.5). O programa pode ser executado com o método `Program.run()` e as informações de execuções específicas de um determinado programa são extraídas com a classe auxiliar `Run`.

Project

Tipo de classe: principal **Módulo:** `src.main`

A classe `Project` é uma classe principal que representa um objeto de alto nível, que agrega diversas fontes de proveniência de um determinado projeto. Isso inclui as amostras (classe `Sample`), arquivos (classe `File`) e programas (classe `Program`). Com isso, o objeto `Project` segue uma estrutura hierárquica (Figura 3.4) que é serializável no formato JSON. Essa funcionalidade de serialização permita que o projeto se atualize e escreva as mudanças em disco durante o tempo de execução. Para ativá-la, é necessário configurar o atributo `Project.auto_update`. Assumindo o estado de código do Snippet 3.2:

```
# inserir projeto no banco
project.update_db()

# configurar atributo
project.auto_update = True

# ao rodar o programa na segunda amostra, o banco atualiza
   automaticamente
with project["sample_2"] as sample:
    sample.add_programs(prodigal(sample))
    sample.run_programs()
```

Snippet 3.7: Atualizando projetos em tempo de execução com `auto_update`.

Nesse exemplo, uma vez que o atributo `auto_update` é ativado, quando programas são adicionados ou executados no projeto, a entrada do projeto no banco de dados BioProv é automaticamente atualizada. Embora encapsulem toda a informação de um deter-

minado workflow, os objetos criados pela classe `Project` são modelados como entidades de proveniência pela classe PROVEntity.

Run

Tipo de classe: auxiliar **Módulo:** `src.main`

A classe `Run` é uma classe auxiliar que coleta informações sobre a execução de um programa. Quando o método `Program.run()` é executado, um objeto `Run` é criado e adicionado ao atributo `Program.runs`. O objeto `Run` possui diversos atributos com informações de proveniência da tarefa que foi executada, como `Run.cmd`, `Run.start_time`, `Run.end_time`, `Run.stdout`, `Run.stderr`, entre outros. Esse objeto também registra o ambiente computacional na qual a tarefa foi executada, o que depois será modelado como um agente de proveniência.

Sample

Tipo de classe: principal **Módulo:** `src.main`

A classe `Sample` é uma classe principal que descreve uma amostra biológica. É modelada como uma entidade de proveniência pela classe PROVEntity. Os objetos criados por esta classe, assim como objetos da classe `Project`, podem ter arquivos e programas associados a eles. Se o conteúdo de um determinado arquivo se referir somente à dados de uma determinada amostra, ele deve ser associado à essa amostra. Programas associados à uma determinada amostra podem ser construídos com base nos arquivos que estão associados à mesma, tanto no sentido de usar arquivos como *input* quanto criar arquivos de *output* nomeados com base no nome da amostra ou dos arquivos de *input*. Amostras também podem ter atributos arbitrários que podem ser adicionados ao seu registro de proveniência.

SeqFile

Tipo de classe: auxiliar **Módulo:** `src.files`

A classe `SeqFile` é uma classe auxiliar que herda da classe `File`. Possui interface com a biblioteca BioPython, especificamente com os módulos SeqIO e AlignIO. Isso permite que ao importar um arquivo com esta classe, dados de domínio referente às sequências biológicas contidas nos arquivos são importados como atributos do objeto criado pela classe. Para importar tais dados, é necessário especificar o formato de arquivo (com o argumento `format`), e/ou o parser (com o argument `parser`), e ativar o argumento `import_records`:

```
import bioprov as bp
from bioprov.data import synechococcus_genome
```

```
myAssembly = bp.SeqFile(synechococcus_genome,
                        tag="assembly",
                        format="fasta",
                        parser="seq",
                        import_records=True)

# supondo um arquivo de alinhamento no formato clustal
myAlignment = bp.SeqFile("Synechococcus.aln",
                         tag="alignment",
                         format="clustal",
                         parser="align",
                         import_records=True)
```

Snippet 3.8: Importando dados de domínio com a classe `SeqFile`.

No código acima, as sequências podem ser acessadas pelos seus identificadores através do atributo `SeqFile.records[id-da-sequencia]`. Os objetos de sequência são construídos pela classe SeqRecord ou por alguma das classes do módulo Align, ambas da biblioteca BioPython. Isso propicia uma diversidade de operações que podem ser realizadas nesses objetos.

SeqStats

Tipo de classe: auxiliar **Módulo:** `src.files`

A classe `SeqStats`[14] é uma classe auxiliar que contém os dados de domínio de um objeto construído com a classe `SeqFile`. É implementada como uma *data class* do Python[13]. É invocada através do método `SeqFile._calculate_seqstats`. Especificamente, contém os seguintes metadados:

Tabela 3.8: Métricas de sequência que compõe a classe `SeqStats`.

Atributo	Tipo	Descrição
`number_seqs`	`int`	número total de sequências
`total_bps`	`int`	número total de nucleotídeos ou aminoácidos
`mean_bp`	`float`	comprimento médio das sequências
`min_bp`	`int`	comprimento da menor sequência
`max_bp`	`int`	comprimento da maior sequência
`N50`	`int`	métrica de N50[15]
`GC`	`float`	conteúdo GC[16] (somente para arquivos de nucleotídeos)

Ter rápido acesso a essas métricas pode ser incrivelmente útil para compreender os resultados de um workflow. Por exemplo, o comprimento médio das sequências e N50 servem para avaliar a qualidade da montagem de um genoma. O total de sequências em um determinado arquivo pode servir para determinar a eficácia de um processo de anotação das sequências através de alinhamento contra um determinado banco de dados.

Step

Tipo de classe: auxiliar **Módulo:** `src.workflow`

A classe `Step` é uma classe auxiliar que herda da classe `PresetProgram`. Os objetos criados por ela são programas especializados que usados para construir presets de *workflows* (com a classe `Workflow`). Possuem os atributo adicionais `default` e `kind`, que especificam respectivamente se o programa, por padrão, é executado ou não, e se o programa deve ser associado a uma amostra ou ao projeto.

Workflow

Tipo de classe: auxiliar **Módulo:** `src.workflow`

A classe `Workflow` é uma classe auxiliar que é usada para criar presets de workflow. Um preset de workflow será uma sequência de programas (construídos com a classe `Step`) que serão executada em um conjunto de amostras e/ou para um projeto. *Workflows* podem ser executados usando a aplicação CLI da biblioteca, com o comando `bioprov <nome_do_workflow>`. O *workflow* cria um objeto da classe `Project` referente à aquela determinada execução. Presets de *workflows* devem ser armazenados no subpacote `workflows`, similar ao subpacote `programs`.

WorkflowOptionsParser

Tipo de classe: auxiliar **Módulo:** `workflows.workflows`

A classe `WorkflowOptionsParser` é uma classe auxiliar que tem a finalidade de gerenciar a aplicação CLI no que se refere aos argumentos que são passados para cada workflow. É invocada pelo módulo `bioprov.bioprov`. Para um workflow ser disponível através da aplicação CLI, é necessário criar um método estático nessa classe que invoque o *workflow* e os passos a serem executados.

```
class WorkflowOptionsParser:
    """
    Class for parsing command-line options.
    """

    # [...]

    @staticmethod
    def _genome_annotation(kwargs, steps):
        """
        Runs genome annotation workflow
        :return:
        """
```

```
        main = genome_annotation(**kwargs)
        main.run_steps(steps)
```

Snippet 3.9: Exemplo de método estático pela class `WorkflowOptionsParser`.

3.2.3 Compatibilidade com modelo W3C-PROV

A modelagem adotada por BioProv (descrita na Seção 3.1.1 e representada na Figura 3.4) procura englobar os principais elementos presentes em um *workflow* de bioinformática. Isso inclui as amostras, arquivos, programas usuários(as) e ambientes envolvidos na execução do workflow. Na implementação de BioProv, o objeto que contém essa informação é serializado em formato JSON, mas essa modelagem também pode ser representada de outras formas, para inserção em um banco relacional, por exemplo[19]. O diagrama de classes dessa modelagem é ilustrado na Figura 3.9. É importante levar em conta que os objetos `File` e `Program` podem ser substituídos por classes-filhas, como `SeqFile` ou `Directory` (para `File`) ou `PresetProgram` (para `Program`).

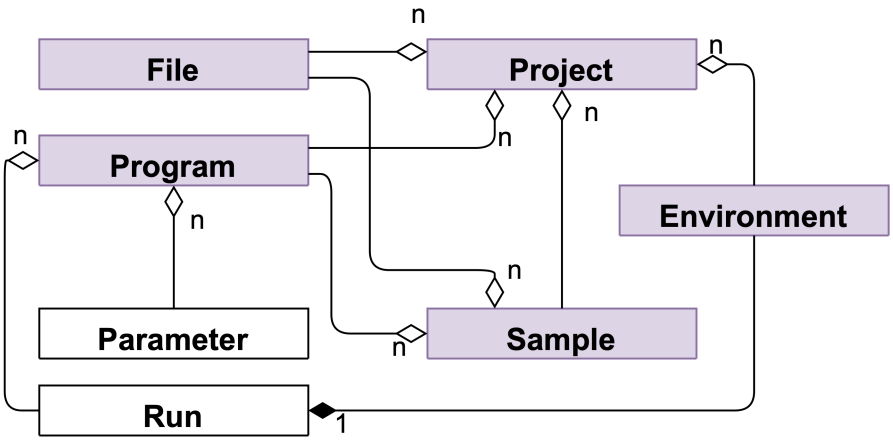


Figura 3.9: Diagrama de classes dos objetos BioProv que representam elementos de um *workflow* de bioinformática. As classes principais estão em roxo e as auxiliares, em branco (discussão na Seção 3.1.1).

Uma vez que a proveniência do *workflow* foi registrada dessa forma, é possível projetar as relações dessa modelagem para os elementos de proveniência. Para isso, os documentos de proveniência criados com BioProv se baseiam na biblioteca PROV, que garante a compatibilidade com o padrão W3C-PROV (para uma descrição detalhada consultar a Subseção 2.3.1). A interface entre as duas bibliotecas é realizada pelos métodos internos da classe `BioProvDocument`. A Tabela 3.2 ilustra a correspondência entre objetos BioProv e elementos de proveniência PROV. O objeto `BioProvDocument` contém o atributo `PROVDocument`, que é uma instância da classe PROVDocument da biblioteca PROV, e logo possui todas as propriedades dessa classe. Um determinado documento irá conter *bundles* associados, que por sua vez são "subdocumentos" usados para agrupar e descrever

[19] O SGBD interno de BioProv é orientado a documentos.

elementos de proveniência do conjunto maior (para uma descrição detalhada e exemplos, consulte a Seção 2.3.1 ou a documentação do pacote). Nos documentos gerados por BioProv, um bundle é declarado para o projeto, e são declarados um para cada amostra e um para cada usuário (Tabela 3.9).

Tabela 3.9: Documento e bundles definidos em um projeto BioProv.

Representação	Descrição	Classe BioProv	Classe PROV
-	Documento de proveniência de alto nível contendo todos os bundles	`BioProvDocument`	`PROVDocument`
project:<project.tag>	Bundle descrevendo o projeto com arquivos e programas associados	`Project`	`PROVBundle`
users:<user.name>	Bundle descrevendo um usuário com ambientes associados	`Environment`	`PROVBundle`
samples:<sample.name>	Bundle descrevendo a amostra com arquivos e programas associados	`Sample`	`PROVBundle`

Quando o objeto `BioProvDocument` é criado, por padrão, seu método construtor[20] invoca métodos privados da classe que criam os elementos de proveniência e os inserem no projeto. Esses métodos podem ser chamados separadamente, mas o recomendado é que sejam chamados durante a execução do método construtor. Dois parâmetros do método construtor ajudam a definir a granularidade das informações que serão inseridas no documento de proveniência: `add_attributes` e `add_users`. O primeiro adiciona os atributos dos objetos BioProv como atributos de proveniência da respectiva entidade, atividade ou agente. O segundo é uma opção que permite omitir a criação de agentes de proveniência representando os usuários e seus respectivos ambientes computacionais, o que pode ser útil para criar documentos (especialmente visualizações/gráficos) mais sucintos.

Uma vez criado o documento de proveniência, é possível exportá-lo de diferentes formas. Alguns exemplos são o formato PROV-N, o formato gráfico DOT[13] (Figura 3.10), e o upload do do documento para a plataforma ProvStore. ProvStore é um serviço web livre para armazenar, visualizar e colaborar em documentos de proveniência. Para poder fazer uso dessa funcionalidade de BioProv, o usuário deve criar uma conta na plataforma, configurar suas credenciais (usando a aplicação CLI da biblioteca), e exportar o objeto usando o método `BioProvDocument.upload_to_provstore()`. Uma vez armazenado na plataforma, o documento pode ser visualizado e exportado de diversas maneiras (Figura 3.11). Outras formatos de exportação também são disponíveis através da classe PROVDocument, como PROV-RDF, PROV-XML e PROV-JSON.

3.2.4 Integração com dependências

Como mencionado anteriormente (Seção 3.2), BioProv faz uso de algumas dependências (Tabela 3.4) para o seu funcionamento. As três principais dependências de BioProv são as bibliotecas PROV, BioPython, e TinyDB, que implementam, respectivamente, a compatibilidade com o modelo W3C-PROV, os parsers para formatos biológicos, e o banco

[20] Em Python, o método construtor de uma classe, ou seja, a função da classe que instancia um objeto, é definido pela função `Classe.__init__()`, similarmente aos pacotes que são inicializados pelo módulo `__init__.py`.

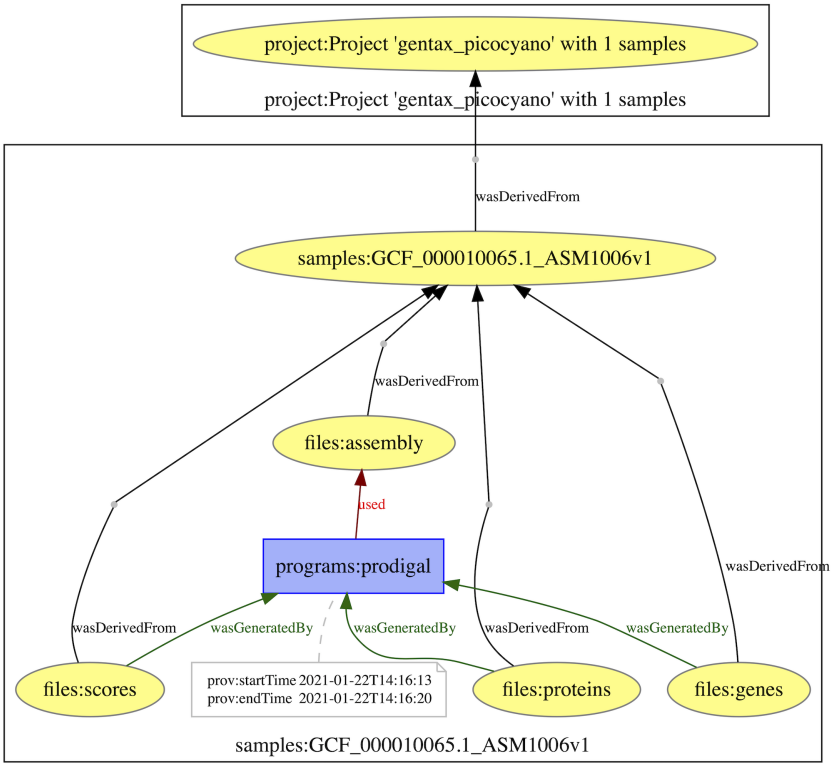


Figura 3.10: Exemplo de uma visualização gerada a partir de um documento de proveniência BioProv.

de dados interno da aplicação. Para isolar essas funcionalidades, elas são restritas, também respectivamente em relação à cada biblioteca, aos módulos `src.prov`, `src.files` e `src.config`. As dependências de BioProv são declaradas no módulo `setup.py`, na raíz do pacote, como usualmente é feito para pacotes Python. A distribuição do Bioconda possui a dependência adicional do programa Prodigal, que é usado para testar o programa com Pytest. As dependências para rodar presets de programas ou de *workflows*, como o preset do programa Diamond ou o preset de *workflow* "Kaiju" não vem distribuídas juntamente de BioProv, sendo necessária a instalação separada, o que é facilmente feito com o gerenciador `conda` (assim como BioProv, esses pacotes estão disponíveis através do Bioconda).

3.2.5 Contribuições externas

BioProv é um pacote de *software* livre, sob a licença MIT[21], e portanto é aberto para contribuições externas. No dia 16 de outubro de 2020, foi feita a primeira contribuição para a biblioteca, pelo usuário @jvfe, que melhorou alguns aspectos do arquivo README e adicionou um guia de contribuição para o repositório. Desde então, o mesmo usuário

[21] `https://choosealicense.com/licenses/mit/`

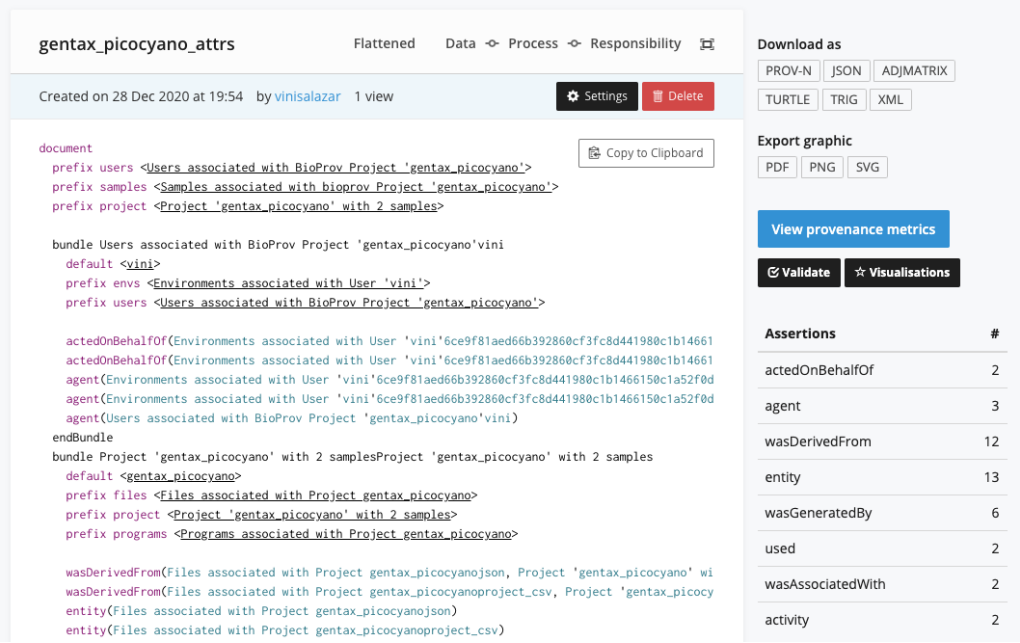


Figura 3.11: Documento gerado por BioProv e armazenado na plataforma ProvStore usando a API disponível na biblioteca.

contribuiu com outras pequenas funcionalidades, especialmente com a adição de mais presets de programas no subpacote `programs`, como os programas MAFFT, Muscle, Kallisto e outros. Um sumário dos contribuidores pode ser encontrado nessa página.

3.3 Sumário

Foi apresentada a biblioteca BioProv para criação de documentos de proveniência em *workflows* de bioinformática. BioProv é baseada em outras bibliotecas Python de código aberto e é implementada conforme boas práticas de engenharia de *software*. O funcionamento de BioProv envolve a criação de objetos que modelam os elementos de um WFB e automaticamente extraem atributos e relações de proveniência. Essa informação pode ser serializada em formato JSON, CSV ou inserida no banco de dados interno da aplicação. BioProv apresenta suporte, através de dependências, para uma variedade de formatos de arquivos biológicos, e também implementa presets próprios para a realização de operações comuns, como busca por alinhamento ou alinhamento múltiplo de sequências. A página de documentação apresenta uma referência completa da API BioProv, bem como tutoriais e exemplos de uso. Usuários são encorajados a contribuir seus próprios presets e *workflows* para a biblioteca, de acordo com o guia de contribuição e a licença. Os projetos monitorados com BioProv também podem ser facilmente armazenados na web através do serviço livre ProvStore. Essas funcionalidades consolidam BioProv como uma adição valiosa para o ecossistema de *software* de *workflows* bioinformáticos.

Capítulo 4

Avaliação experimental

Com o objetivo de avaliar o desempenho da biblioteca BioProv, foi realizado um estudo de caso de um WFB de taxonomia genômica microbiana. Tal estudo serve como um exemplo real de uso da biblioteca, e é usado para ilustrar as capacidades totais da mesma. Diferentemente dos exemplos reproduzidos anteriormente, no Capítulo 3, a presente implementação é um estudo de caso real, e portanto não tem fins didáticos e utiliza as funções mais avançadas da biblioteca. Foi possível descrever o tipo de proveniência coletado, a carga computacional implicada e que tipos de dados de domínio são coletados por BioProv. O estudo de caso também serve como referência para criação de futuros *workflows*.

4.1 Ambiente computacional

Os experimentos foram executados no servidor Prometheus do Laboratório de Microbiologia, Instituto de Biociências da UFRJ. A Prometheus é uma máquina com dois Intel Xeon 6230 2.1 GHz de 20 cores, totalizando 40 processadores, com tecnologia Hyper-Threading (HT) e Turbo Boost até 3.9 GHz. Possui 12 memórias de 32GB DDR4, totalizando 384 GB de RAM. Para armazenamento, são dois discos de estado sólido de 1.92 TB Micron 5200 PRO, com interface SATA de 6 Gb/s, onde fica o sistema operacional Ubuntu 18.04LTS e o diretório `/tmp`, e 2 discos rígidos de 2 TB Seagate Exos 7E200 SATA, que são usados para montar o diretório `/home`, onde os experimentos foram executados.

BioProv foi instalada usando `conda` (ver detalhes na Seção 3.2), juntamente de um script Python[3] contendo a lógica do workflow. O SGBD interno da biblioteca inicia um banco de dados no diretório de instalação, dentro do ambiente `conda`. As consultas podem ser realizadas durante e/ou após o experimento, ao carregar o projeto a partir do banco de dados.

4.2 Estudo de caso: *workflow* de taxonomia genômica

O campo da taxonomia genômica microbiana vem se desenvolvendo intensamente, principalmente por conta do grande aumento na disponibilidade de sequências biológicas de alta qualidade em bancos de dados públicos[1][92, 93]. Estudos recentes apresentam quadros metodológicos [94, 95] para a criação de uma taxonomia microbiana baseada em sequências de genomas completos, no lugar de caracteres morfológicos ou análises de genes marcadores, a chamada "taxonomia polifásica", que predominava desde a década de 70 [96]. O paradigma atual de taxonomia microbiana coloca as sequências de genoma completo como o *gold standard* de material tipo para descrição taxonômica, e propõe análises de distância genética (como AAI e ANI[2]) para definir um limite entre espécies biológicas [98–100].

Um grupo de particular interesse para a taxonomia microbiana são as picocianobactérias, especialmente dos gêneros *Synechococcus* e *Prochlorococcus*, devido a sua importância ecológica [101], biotecnológica [102] e por sua diversidade genética [103]. As bactérias desse grupo são os procariotos fotossintetizantes mais abundantes do planeta, sendo responsáveis por até um quarto da produção primária global [104]. Apesar disso, seu status taxonômico continua em discussão. Estudos de taxonomia genômica dividem *Prochlorococcus* e *Synechococcus* em diferentes gêneros [94, 105–108], e a recente disponibilidade de um grande volumes de genomas para esses gêneros [109] levantou a importância de realizar-se novas análises taxonômicas.

O presente estudo de caso é uma adaptação da metodologia utilizada para analisar a taxonomia genômica de *Prochlorococcus* e *Synechococcus*, cujo os resultados foram publicados separadamente para cada grupo (TSCHOEKE *et al.* [110], SALAZAR *et al.* [111]). O desenvolvimento dessa metodologia motivou a criação da biblioteca BioProv; a coleta de dados de proveniência é particularmente importante no campo da taxonomia, pois a criação de documentos de proveniência para análises taxonômicas reforça a formalização das descrições de táxons discriminados por tais análises, e permite a prática de uma "sistemática guiada por bancos de dados", um conceito proeminente entre taxonomistas [112, 113], mas que ainda não foi amplamente adotado.

Nesse cenário, estabelecer um *workflow* de bioinformática descrito no formato W3C-PROV torna-se relevante, pois este representa um registro da análise genômica utilizada para a determinação e/ou descrição taxonômica de bactérias. Os diferentes formatos de representação do modelo W3C-PROV o tornam legível tanto para humanos (formato PROV-N) quanto máquinas (formatos PROV-JSON, PROV-XML, PROV-RDF), e as funci-

[1] Para genomas bacterianos, isso também foi impulsionado pelos avanços em algoritmos que permitem a reconstrução de genomas a partir de metagenomas, no processo denominando *genome binning* [49, 90, 91].

[2] *Average amino acid identity* e *average nucleotide identity*, respectivamente, são medidas de distância entre genomas completos que se referem a similaridade média de amino ácidos e de nucleotídeos.[97]

onalidades da biblioteca BioProv permitem o armazenamento do *workflow* tanto em disco (através do banco de dados interno da biblioteca) quanto na Web (através do uso da API do serviço ProvStore).

Conjunto de dados e instrumentação

O presente estudo de casos utiliza um subconjunto reduzido de amostras do estudo SALAZAR *et al.* [111] (que originalmente conta com mais de mil amostras), apresentando execuções para 10, 50, e 100 amostras. As amostras são sequências de genoma completo depositadas no banco Genbank [114], e o ponto de entrada do *workflow* é um único arquivo de texto contendo os códigos de acesso de cada genoma.

Para realizar a instrumentação do *workflow* de taxonomia genômica com a biblioteca BioProv, foi desenvolvido um script Python[3] com 14 funções que faz download dos dados, cria um projeto BioProv, e roda a análise para os respectivos dados.

4.2.1 Tarefas

O *workflow* de taxonomia genômica é dividida em duas etapas, a etapa inicial, de **pré-processamento**, na qual é realizado o download dos dados e a formatação do arquivo de metadados que cria o projeto BioProv, e a etapa de **processamento**, que executa as tarefas principais do workflow. Para fins de medição de tempo de execução (discutida na Subseção 4.2.4), serão considerados somente as tarefas da etapa de processamento, tendo em vista que o tempo download dos dados é dependente da velocidade de conexão da internet. No entanto, é importante notar que, mesmo sendo executadas antes da criação do projeto BioProv, as tarefas da etapa de pré-processamento são posteriormente associadas ao projeto, ou seja, a proveniência dessas tarefas e dos arquivos envolvidos também é monitorada.

Os passos de cada etapa estão ilustrados na Tabela 4.1. As tarefas principais do *workflow* incluem a predição de genes com Prodigal, que é feita para cada amostra (um exemplo de uma tarefa executada em batch mode[10]); a comparação pareada de arquivos contendo sequências codificantes de cada amostra[4], realizada pelo programa `fastani` [115]; a formatação do arquivo resultando do `fastani` em uma tabela simétrica de dissimilaridade, e a clusterização hierárquica através da distância de Manhattan[5], que então é plotada como um dendrograma.

No *workflow* apresentado, as tarefas do programas `prodigal` e `fastani` são realizadas por *software* de terceiros (*i.e. third-party software*), e os algoritmos implementados por esses programas são bem detalhados nas publicações correspondentes [80, 115]. As ta-

[3] https://github.com/vinisalazar/BioProv/blob/use-case/use-case/workflow.py

[4] É obtido o valor de ANI^2 entre cada amostra.

[5] https://docs.scipy.org/doc/scipy/reference/generated/scipy.spatial.distance.cityblock.html

Tabela 4.1: Programas executados no *workflow* de taxonomia genômica.

Etapa	Nome	Função
Pré-processamento	`ncbi-genome-download`	Download dos dados do GenBank.
	`sort`	Deduplicar a tabela de metadados que foi obtida.
	`gunzip`	Descomprimir arquivos obtidos.
	`sed_gz`	Substituir caminhos de arquivo na tabela de metadados.
	`sed_column`	Substituir nome de coluna na tabela de metadados.
Processamento	`prodigal`	Predição de genes. Executado para **cada amostra**.
	`fastani`	Comparação pareada entre o arquivo de genes de cada amostra.
	`format_fastani_output`	Formatar saída do programa `fastani`.
	`cluster_and_plot`	Realizar clusterização hierárquica e plotar dendrograma.

refas seguintes, de `format_fastani_output` e `cluster_and_plot` são implementadas em scripts Python customizados.

Criação da matriz de distância

A primeira dessas tarefas é uma transformação do arquivo de saída do `fastani`; tal arquivo é uma tabela com N^2 linhas e 5 colunas, onde N é o número de amostras (denominadas n na Tabela 4.2). Essa tabela é transformada com a função `pd.pivot_table()` da biblioteca Pandas, para criar uma matriz com dimensões nxn, onde cada linha e coluna é correspondente a um genoma. Como a comparação realizada pelo programa `fastani` não é comutativa, ou seja, a ordem do par de genomas a serem comparados importa, a matriz não é simétrica. No entanto, essa diferença é desprezível, e com fins de criar uma matriz de distância simétrica, a média entre os valores das duas comparações é calculada. Por exemplo, assumindo a matriz D e os genomas n_i e n_j, para cada valor de ANI $D_{i,j}$ e $D_{j,i}$ na matriz, será calculada a média $\mu_{i,j} = \frac{D_{i,j}+D_{j,i}}{2}$, e ambos os valores $D_{i,j}$ e $D_{j,i}$ são transformados no valor $\mu_{i,j}$. Como a intenção é criar uma matriz de dissimilaridade entre os genomas, a matriz D sofre a transformação $|D - 100|$, especificada pelo parâmetro `-invert` do script `format_fastani_output.py`.

Tabela 4.2: Exemplo da matriz de dissimilaridade de ANI resultante depois da formatação com o script `format_fastani_output.py`, onde todos os valores $\mu_{i,j} = \mu_{j,i}$ para todos os valores de i e j e quando $i = j \Rightarrow D_{i,j} = 0.0$.

	n_1	n_2	...	n_i
n_1	0.0	$\lvert\mu_{1,2} - 100\rvert$	...	$\lvert\mu_{1,i} - 100\rvert$
n_2	$\lvert\mu_{2,1} - 100\rvert$	0.0	...	$\lvert\mu_{2,i} - 100\rvert$
...	...	...	...	...
n_j	$\lvert\mu_{j,1} - 100\rvert$	$\lvert\mu_{j,2} - 100\rvert$	...	0.0

Clusterização hierárquica e visualização

A tarefa subsequente, executada pelo script `hierarchical_clustering.py`, tem como entrada um arquivo como o da Tabela 4.2, gerada pelo programa anterior. Esse script executa três passos:

1. conversão da matriz de distância em um vetor;
2. clusterização hierárquica do vetor de distância;
3. plotagem do resultado da clusterização em um dendrograma.

O primeiro passo, da conversão da matriz de distância em um formato vetorial de distância, também referido como "matriz condensada de distância", é feito através da função `squareform()`[6]. A função executa a seguinte operação:

Dada uma matriz D de distância quadrada e simétrica, com dimensões nxn, a operação $X =$ `squareform(D)` retorna um vetor X de tamanho $\binom{n}{2}$ onde $X[\binom{n}{2} - \binom{n-1}{2} + (j - i - 1)]$ é a distância entre os pontos i e j.

No segundo passo, o vetor X é clusterizado com a função `linkage()`[7]. O método de clusterização utilizado é o ''`complete`'', conhecido como "*Farthest Point Algorithm*" [116]. Esse método computa a distância $d(s,t)$ entre os clusters s e t, combinando s e t em um cluster u. O algoritmo é inicializado com cada cluster sendo assinalado para uma observação do vetor X e então as distâncias são computadas iterativamente através da fórmula:

$$d(u,v) = \max(dist(u[i], v[j]))$$

Onde *dist(u[i], v[j])* é dado pela distância de Manhattan $\sum_i |u_i v_i|$ para os clusters u e v (sendo u a combinação dos cluster iniciais s e t e v qualquer cluster remanescente que não foi computado). O resultado vem codificado como uma matriz Z de clusterização que é usada para gerar um dendrograma.

4.2.2 Saídas

O *workflow* de taxonomia genômica inicia a partir de um único arquivo de texto contendo números de acesso de cada amostra a ser baixada. Na etapa de pré-processamento, é criado o arquivo de metadados, que vai importar as amostras como um projeto BioProv, e é feito o download dos arquivos a serem processados (especificamente, as sequências de genoma completo, identificados pela tag "genome_assembly").

[6] https://docs.scipy.org/doc/scipy/reference/generated/scipy.spatial.distance.squareform.html

[7] https://docs.scipy.org/doc/scipy/reference/generated/scipy.cluster.hierarchy.linkage.html

Após essa etapa, são criados seis arquivos a nível de projeto[8], e, para cada amostra, são criados dois arquivos, contendo as sequências codificantes dos genomas como nucleotídeos (arquivo de genes) ou aminoácidos (arquivo de proteínas).

Tabela 4.3: Arquivos de saída do *workflow* de taxonomia genômica.

Etapa	Tag do arquivo	Conteúdo
Pré-processamento	`project_csv`	Arquivo tabular contendo metadados das amostras e o caminho de cada arquivo de sequência genômica.
	`genome_assembly`	Caminho do arquivo de genoma completo de cada amostra.
Processamento	`genes`	Arquivo contendo sequências codificantes de nucleotídeos de cada amostra.
	`proteins`	Arquivo contendo sequências codificantes de aminoácido de cada amostra.
	`fastani_input`	Arquivo de texto contendo o caminho de todos os arquivos "`genes`".
	`fastani_output`	Tabela com comparações pareadas entre todos os genomas.
	`fastani_output_fmt`	Tabela anterior formatada em uma matriz simétrica de distância.
	`labels`	Arquivo usado para renomear os ramos da figura de dendrograma.
	`dendrogram`	Figura de dendrograma ilustrando as distâncias genéticas entre as amostras.

Tais arquivos se diferem do arquivo da sequência genômica pelo fato de conterem apenas os trechos codificantes da sequência genômica. Enquanto o arquivo da sequência genômica possui de uma até algumas dúzias de sequências contínuas, denominados *contigs*, os arquivos ‘‘`genes`’’ e ‘‘`proteins`’’ vão possuir entre dois a cinco mil sequências menores[9], que correspondem respectivamente as sequências de cada gene ou cada proteína transcritas pelo genoma. O uso de tais arquivos para a comparação através de métricas como ANI ou AAI é recomendado pois aumenta a sensibilidade da análise [115].

Os arquivos então são concatenados em um único arquivo ‘‘`fastani_input`’’. Esse passo demonstra uma característica importante da biblioteca BioPython: é possível utilizar a sintaxe nativa de Python para realizar operações *ad hoc* durante a execução do código. Isso é extremamente conveniente, pois as funcionalidades da linguagem podem interagir com os objetos BioProv. É importante notar que essas operações não ficam guardadas como um objeto `Program`; logo, sua proveniência não é armazenada. No entanto, como esse exemplo ilustra, utilizam-se as funções *built-in* Python para criar o arquivo, que então é convertido no objeto `File` e adicionado ao projeto. Logo, a proveniência do arquivo é armazenada, mas não a do programa que o gerou. No Snippet 4.1, são criados dois objetos `File`, sendo que o conteúdo do objeto `fastani_input` é escrito pela função

[8]Isso é, arquivos associados a zero ou mais de uma amostras. Nesse caso, são arquivos que contém informação de todas as amostras do projeto. Ver a discussão na Seção 3.2.

[9]O número de genes está linearmente relacionado com o tamanho do genoma. No conjunto de dados em questão, o comprimento dos genomas varia entre 1.5 até 5 megabases.

usando as funções de IO nativas do Python (sua proveniência não é armazenada como a de um `Program`). O objeto `fastani_output` vai servir como arquivo de saída para o programa `fastani`, que é definido posteriormente.

```
import bioprov as bp

def fastani(proj):
    fastani_input = bp.File(f"data/fastani_input_{proj.tag}.
   txt", tag="fastani_input")
    fastani_output = bp.File(
        f"data/fastani_output_{proj.tag}.txt", tag="
   fastani_output"
    )
    proj.add_files(fastani_input)
    proj.add_files(fastani_output)

    # writing with native Python IO functions
    with open(proj.files["fastani_input"].path, "w") as f:
        for file in (sample.files["genome_assembly"] for
   sample in proj):
            f.write(str(file) + "\n")

    _fastani = bp.Program("fastani")

    params = [
        bp.Parameter("--refList", proj.files["fastani_input"
   ], kind="input"),
        bp.Parameter("--queryList", proj.files["
   fastani_input"]),
        bp.Parameter("--threads", bp.config.threads),
        bp.Parameter("--fragLen", 200),
        bp.Parameter("--minFraction", 0.01),
        bp.Parameter("--output", proj.files["fastani_output"
   ], kind="output"),
    ]

    for param in params:
        _fastani.add_parameter(param)

    proj.add_programs(_fastani)
    _fastani.run(suppress_stderr=False)
```

Snippet 4.1: É possível usar as funcionalidades da linguagem Python para realizar operações durante a execução do código BioProv.

O programa `fastani` tem como entrada o arquivo ‘‘`fastani_input`’’ e realiza a comparação pareada entre os arquivos de genes de cada amostra. O arquivo de saída é atribuído ao objeto `File` que já está associado ao projeto. A tabela resultante, ‘‘`fastani_output`’’ é formatada com o script `format_fastani_output.py` (referir à 4.2.1), criando o arquivo ‘‘`format_fastani_output`’’, que contém a matriz simétrica de distância. Tal matriz é o arquivo de entrada para a tarefa final, que é a clusterização e plotagem com o script `hierarchical_clustering.py`, gerando o arquivo final do workflow, a figura do dendrograma, identificada como ‘‘`dendrogram`’’.

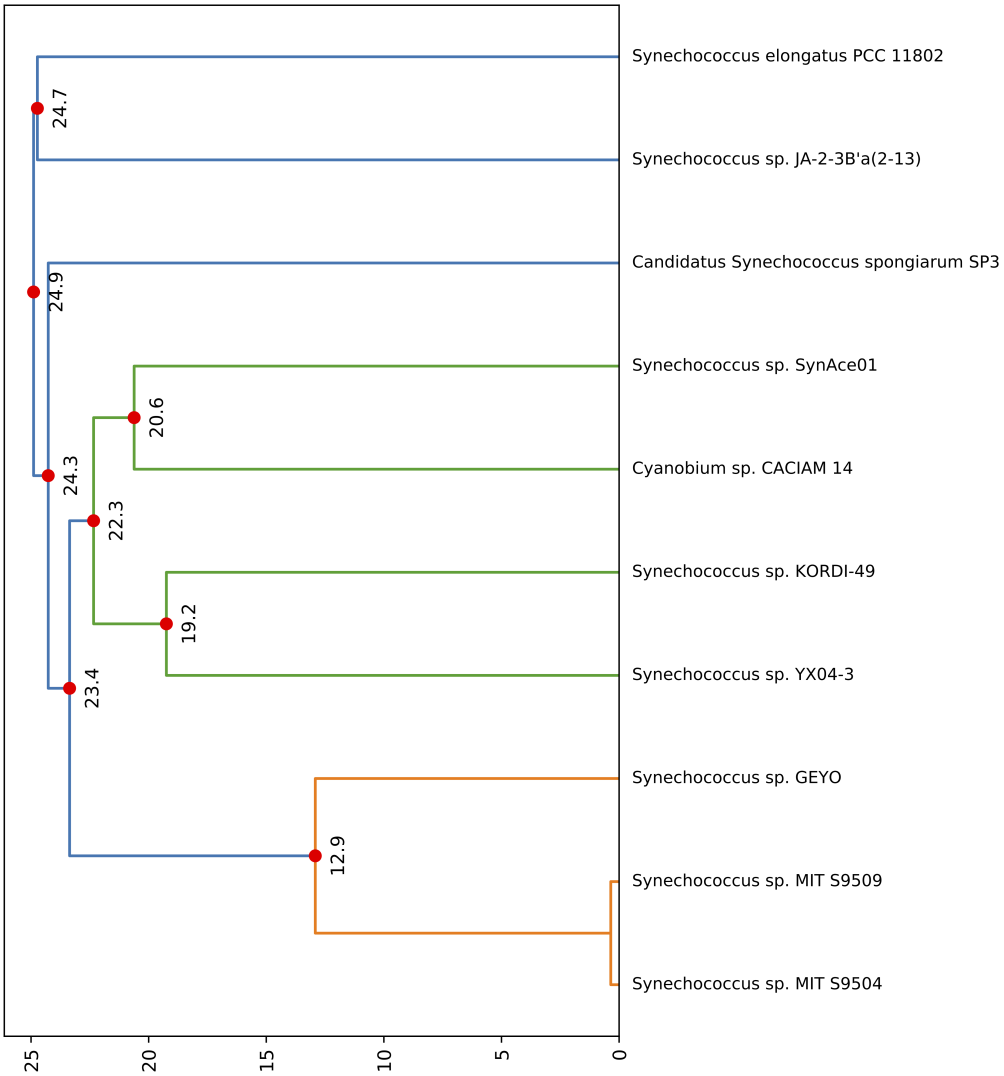


Figura 4.1: Exemplo do dendrograma criado pelo *workflow* de taxonomia genômica usando um conjunto de dados de 10 amostras. Cada ramo corresponde a uma amostra biológica e cada nodo indica a distância genética máxima ($|100 - ANI|$) entre as duas subárvores.

A Figura 4.1 ilustra um exemplo gerado para 10 amostras. O valor de cada nodo identifica a distância genética entre os ramos filhos daquele nodo. Para a métrica ANI, distâncias maiores que 5 podem ser consideradas espécies diferentes e valores maiores que 17 podem ser considerados gêneros diferentes [97, 115]. Essa figura é um exemplo de visualização criada por um *workflow* BioProv que tem relevância tanto para relatórios preliminares quanto para análises mais detalhadas de taxonomia genômica.

4.2.3 Proveniência

Os dados de proveniência coletados por BioProv podem ser divididos em duas categorias principais: **atributos**, que são propriedades descritivas dos elementos envolvidos no workflow, e **relacionamentos**, que descrevem as relações de proveniência entre os elementos do workflow. Para compreender como esses dados são coletados, é importante recapitular os principais objetos BioProv (Tabela 3.2). Ao final da execução do workflow, temos um objeto `Project` que contém os outros objetos BioProv que representam amostras, arquivos e programas. Tal objeto é armazenado no banco de dados interno da biblioteca e é identificado pelo atributo `Project.tag`. A melhor forma de consultar os dados de proveniência do *workflow* é carregar esse objeto em um interpretador Python e chamar seus atributos.

Consultas no ambiente Jupyter

Abaixo estão demonstradas algumas consultas que podem ser realizadas em um ambiente Jupyter.

Começa-se por importar o projeto. Para poder consultar certos atributos de arquivos, é necessário ativar o parâmetro `import_records` ao carregar o projeto. Esse parâmetro importa os dados de domínio relativos a cada sequência presente no arquivo. Também é possível importar esses dados individualmente para cada arquivo.

[1]:
```
import bioprov as bp

proj = bp.load_project("genomes_10", import_records=True)
```

Consulta 1 - tempo de execução

- Quanto tempo demorou para rodar o programa Prodigal para cada amostra? As execuções terminaram?

O tempo de duração e o status da execução são identificados respectivamente pelos atributos `duration` e `status`.

[2]:
```
for sample in proj:
    print(sample.name, '\t',
```

```
        sample.programs['prodigal'].duration, '\t',
        sample.programs['prodigal'].status)
```

```
GCA_900473955.1         0:00:16.457727          Finished
GCA_009498715.1         0:00:06.926076          Finished
GCA_003712085.1         0:00:16.376083          Finished
GCA_001885215.1         0:00:14.824035          Finished
GCA_001632105.1         0:00:15.760653          Finished
GCA_001631935.1         0:00:15.664144          Finished
GCA_001007665.1         0:00:12.791492          Finished
GCA_000737575.1         0:00:14.865194          Finished
GCA_000708525.1         0:00:15.950089          Finished
GCA_000013225.1         0:00:08.158993          Finished
```

Alternativamente, podemos consultar o horário de início e término pelos atributos `start_time` e `end_time`:

[3]:
```
for sample in proj:
    print(sample.name,
        sample.programs['prodigal'].start_time,
        sample.programs['prodigal'].end_time)
```

```
GCA_900473955.1 Thu Jan 14 15:08:38 2021 Thu Jan 14 15:08:55 2021
GCA_009498715.1 Thu Jan 14 15:08:55 2021 Thu Jan 14 15:09:02 2021
GCA_003712085.1 Thu Jan 14 15:09:02 2021 Thu Jan 14 15:09:18 2021
GCA_001885215.1 Thu Jan 14 15:09:18 2021 Thu Jan 14 15:09:33 2021
GCA_001632105.1 Thu Jan 14 15:09:33 2021 Thu Jan 14 15:09:49 2021
GCA_001631935.1 Thu Jan 14 15:09:49 2021 Thu Jan 14 15:10:05 2021
GCA_001007665.1 Thu Jan 14 15:10:05 2021 Thu Jan 14 15:10:18 2021
GCA_000737575.1 Thu Jan 14 15:10:18 2021 Thu Jan 14 15:10:33 2021
GCA_000708525.1 Thu Jan 14 15:10:33 2021 Thu Jan 14 15:10:49 2021
GCA_000013225.1 Thu Jan 14 15:10:49 2021 Thu Jan 14 15:10:57 2021
```

Se quisermos ver a duração do programa FastANI, como está associado ao projeto, não é necessário iterar as amostras:

[4]:
```
print(
    proj.programs['fastani'].start_time,
    proj.programs['fastani'].end_time,
    proj.programs['fastani'].duration,
)
```

```
Thu Jan 14 15:10:57 2021 Thu Jan 14 15:13:43 2021 0:02:46.128187
```

Consulta 2:

- Quantos genes foram identificados em cada genoma? E quantas proteínas?

O número de genes corresponde ao número total de sequências no arquivo `'genes'` (e o mesmo para proteínas), e esse atributo é identificado pelo como `number_seqs`.

[5]:
```
for sample in proj:
    print(sample.name, '\t',
          sample.files['genes'].number_seqs, '\t',
          sample.files['proteins'].number_seqs)
```

```
GCA_900473955.1          2674    2674
GCA_009498715.1          2867    2867
GCA_003712085.1          2745    2745
GCA_001885215.1          2897    2897
GCA_001632105.1          3521    3521
GCA_001631935.1          3526    3526
GCA_001007665.1          2356    2356
GCA_000737575.1          2662    2662
GCA_000708525.1          3237    3237
GCA_000013225.1          2844    2844
```

Consulta 3:

- Qual é o conteúdo GC da sequência de genoma completo? E dos genes?

O conteúdo GC é identificado pelo atributo `GC`.

[6]:
```
for sample in proj:
    print(sample.name, '\t',
          sample.files['genome_assembly'].GC, '\t',
          sample.files['genes'].GC)
```

```
GCA_900473955.1          0.56506         0.57131
GCA_009498715.1          0.54793         0.55344
GCA_003712085.1          0.59305         0.60117
GCA_001885215.1          0.63916         0.64646
GCA_001632105.1          0.55434         0.56584
GCA_001631935.1          0.55424         0.56567
GCA_001007665.1          0.60931         0.61579
GCA_000737575.1          0.61368         0.61893
GCA_000708525.1          0.68561         0.6898
GCA_000013225.1          0.5845          0.59262
```

Consulta 4:

- Qual é o identificador e o comprimento da maior proteína em cada amostra?

A maior sequência em um arquivo é identificada com o atributo `max_seq`. O atributo `min_seq` mostra a menor sequência.

[7]:
```
for sample in proj:
    prot = sample.files['proteins']
    print(sample.name, '\t',
          prot.max_seq.id, '  \t',
          len(prot.max_seq.seq))
```

```
GCA_900473955.1         UCNL01000001.1_159      1533
GCA_009498715.1         CP034671.1_2246         2190
GCA_003712085.1         RHLE01000010.1_30       4523
GCA_001885215.1         CP018091.1_35           2200
GCA_001632105.1         LVHT01000004.1_33       3108
GCA_001631935.1         LVHV01000005.1_212      3108
GCA_001007665.1         JXQG01000004.1_36       2471
GCA_000737575.1         CP006270.1_2242         11267
GCA_000708525.1         JMRP01000043.1_14       4254
GCA_000013225.1         CP000240.1_1209         2506
```

Podemos fazer a mesma consulta no arquivo de genes:

[8]:
```
for sample in proj:
    genes = sample.files['genes']
    print(sample.name, '\t',
          genes.max_seq.id, '  \t',
          len(genes.max_seq.seq))
```

```
GCA_900473955.1         UCNL01000001.1_159      4599
GCA_009498715.1         CP034671.1_2246         6570
GCA_003712085.1         RHLE01000010.1_30       13569
GCA_001885215.1         CP018091.1_35           6600
GCA_001632105.1         LVHT01000004.1_33       9324
GCA_001631935.1         LVHV01000005.1_212      9324
GCA_001007665.1         JXQG01000004.1_36       7413
GCA_000737575.1         CP006270.1_2242         33801
GCA_000708525.1         JMRP01000043.1_14       12762
GCA_000013225.1         CP000240.1_1209         7518
```

A partir do conteúdo do objeto `Project`, e dos atributos dos objetos contidos no mesmo (amostras, programas, e arquivos), é possível construir um objeto `BioProvDocument` que infere os relacionamentos de proveniência. Esse objeto contém um documento de proveniência, representado pela classe PROVDocument (da biblioteca PROV), que descreve os relacionamentos de proveniência entre os elementos do workflow.

[9]:
```
prov = bp.BioProvDocument(proj)
```

Se acessarmos o atributo `ProvDocument`, que contém o documento de proveniência, podemos imprimir as relações no formato PROV-N. É importante notar que o documento de proveniência é dividido em subdocumentos denominados *bundles*, como descrito na Subseção 3.2.3 (referir a tabela 3.9. É possível acessar os *bundles* individuais através do atributo `ProvDocument.bundles`:

[10]:
```
prov.ProvDocument.bundles
```

[10]:
```
dict_values([<ProvBundle: users:vini>, <ProvBundle: project:Project␣
↪'genomes_10'
with 10 samples>, <ProvBundle: samples:GCA_900473955.1>, <ProvBundle:
samples:GCA_009498715.1>, <ProvBundle: samples:GCA_003712085.1>,␣
↪<ProvBundle:
samples:GCA_001885215.1>, <ProvBundle: samples:GCA_001632105.1>,␣
↪<ProvBundle:
samples:GCA_001631935.1>, <ProvBundle: samples:GCA_001007665.1>,␣
↪<ProvBundle:
samples:GCA_000737575.1>, <ProvBundle: samples:GCA_000708525.1>,␣
↪<ProvBundle:
samples:GCA_000013225.1>])
```

O atributo `records` de um *bundle* permite inspecionar os registros de proveniência para esse *bundle*. Por exemplo, para o *bundle* do projeto, temos os seguintes elementos (PROVActivity, PROVEntity) e relacionamentos (PROVAssociation, PROVDerivation, PROVGeneration, PROVUsage):

[11]:
```
list(prov.ProvDocument.bundles)[1].records
```

[11]:
```
[<ProvEntity: project:Project 'genomes_10' with 10 samples>,
 <ProvEntity: files:project_csv>,
 <ProvDerivation: (files:project_csv, project:Project 'genomes_10' with 10
samples)>,
 <ProvEntity: files:log>,
```

```
 <ProvDerivation: (files:log, project:Project 'genomes_10' with 10␣
 ↪samples)>,
 <ProvEntity: files:fastani_input>,
 <ProvDerivation: (files:fastani_input, project:Project 'genomes_10' with␣
 ↪10
samples)>,
 <ProvEntity: files:fastani_output>,
 <ProvDerivation: (files:fastani_output, project:Project 'genomes_10'␣
 ↪with 10
samples)>,
 <ProvEntity: files:fastani_output_fmt>,
 <ProvDerivation: (files:fastani_output_fmt, project:Project 'genomes_10'␣
 ↪with
10 samples)>,
 <ProvEntity: files:labels>,
 <ProvDerivation: (files:labels, project:Project 'genomes_10' with 10␣
 ↪samples)>,
 <ProvEntity: files:dendrogram>,
 <ProvDerivation: (files:dendrogram, project:Project 'genomes_10' with 10
samples)>,
 <ProvActivity: programs:ncbi-genome-download>,
 <ProvAssociation: (programs:ncbi-genome-download,
envs:8289108a0e9fd6a8eb01e9fdfb8f0336585bfd1e2317f2b7f9b37fafe0edebc1)>,
 <ProvActivity: programs:sort>,
 <ProvAssociation: (programs:sort,
envs:8289108a0e9fd6a8eb01e9fdfb8f0336585bfd1e2317f2b7f9b37fafe0edebc1)>,
 <ProvActivity: programs:gunzip>,
 <ProvAssociation: (programs:gunzip,
envs:8289108a0e9fd6a8eb01e9fdfb8f0336585bfd1e2317f2b7f9b37fafe0edebc1)>,
 <ProvActivity: programs:sed_gz>,
 <ProvAssociation: (programs:sed_gz,
envs:8289108a0e9fd6a8eb01e9fdfb8f0336585bfd1e2317f2b7f9b37fafe0edebc1)>,
 <ProvActivity: programs:sed_column>,
 <ProvAssociation: (programs:sed_column,
envs:8289108a0e9fd6a8eb01e9fdfb8f0336585bfd1e2317f2b7f9b37fafe0edebc1)>,
 <ProvActivity: programs:fastani>,
 <ProvAssociation: (programs:fastani,
envs:8289108a0e9fd6a8eb01e9fdfb8f0336585bfd1e2317f2b7f9b37fafe0edebc1)>,
 <ProvUsage: (files:fastani_input, programs:fastani)>,
 <ProvGeneration: (files:fastani_output, programs:fastani)>,
 <ProvActivity: programs:format_fastani_output>,
```

```
 <ProvAssociation: (programs:format_fastani_output,
envs:8289108a0e9fd6a8eb01e9fdfb8f0336585bfd1e2317f2b7f9b37fafe0edebc1)>,
 <ProvUsage: (files:fastani_output, programs:format_fastani_output)>,
 <ProvGeneration: (files:fastani_output_fmt, programs:format_fastani_-
output)>,
 <ProvActivity: programs:cluster_and_plot>,
 <ProvAssociation: (programs:cluster_and_plot,
envs:8289108a0e9fd6a8eb01e9fdfb8f0336585bfd1e2317f2b7f9b37fafe0edebc1)>,
 <ProvUsage: (files:fastani_output_fmt, programs:cluster_and_plot)>,
 <ProvGeneration: (files:dendrogram, programs:cluster_and_plot)>]
```

Inspecionando os 5000 primeiros caracteres do registro PROVN do documento completo:

[12]:
```
print(prov.ProvDocument.get_provn()[:5000], "\n\n", "[...]\n\n")
```

```
document
  prefix project <Project 'genomes_10' with 10 samples>
  prefix users <Users associated with BioProv Project 'genomes_10'>
  prefix samples <Samples associated with bioprov Project 'genomes_10'>

  bundle users:vini
    default <vini>
    prefix envs <Environments associated with User 'vini'>
    prefix users <Users associated with BioProv Project 'genomes_10'>

    agent(users:vini)
    agent(envs:
↪8289108a0e9fd6a8eb01e9fdfb8f0336585bfd1e2317f2b7f9b37fafe0edebc1)
    actedOnBehalfOf(envs:
↪8289108a0e9fd6a8eb01e9fdfb8f0336585bfd1e2317f2b7f9b37fa
fe0edebc1, users:vini, -)
  endBundle
  bundle project:Project 'genomes_10' with 10 samples
    default <genomes_10>
    prefix files <Files associated with Project genomes_10>
    prefix project <Project 'genomes_10' with 10 samples>
    prefix programs <Programs associated with Project genomes_10>
    prefix envs <Environments associated with User 'vini'>

    entity(project:Project 'genomes_10' with 10 samples)
    entity(files:project_csv)
```

```
    wasDerivedFrom(files:project_csv, project:Project 'genomes_10' with 10
samples, -, -, -)
    entity(files:log)
    wasDerivedFrom(files:log, project:Project 'genomes_10' with 10␣
 ↪samples, -,
-, -)
    entity(files:fastani_input)
    wasDerivedFrom(files:fastani_input, project:Project 'genomes_10' with␣
 ↪10
samples, -, -, -)
    entity(files:fastani_output)
    wasDerivedFrom(files:fastani_output, project:Project 'genomes_10' with␣
 ↪10
samples, -, -, -)
    entity(files:fastani_output_fmt)
    wasDerivedFrom(files:fastani_output_fmt, project:Project 'genomes_10'␣
 ↪with
10 samples, -, -, -)
    entity(files:labels)
    wasDerivedFrom(files:labels, project:Project 'genomes_10' with 10␣
 ↪samples,
-, -, -)
    entity(files:dendrogram)
    wasDerivedFrom(files:dendrogram, project:Project 'genomes_10' with 10
samples, -, -, -)
    activity(programs:ncbi-genome-download, 2021-01-14T14:56:12,
2021-01-14T14:56:54)
    wasAssociatedWith(programs:ncbi-genome-download,
envs:8289108a0e9fd6a8eb01e9fdfb8f0336585bfd1e2317f2b7f9b37fafe0edebc1, -)
    activity(programs:sort, 2021-01-14T14:56:54, 2021-01-14T14:56:54)
    wasAssociatedWith(programs:sort,
envs:8289108a0e9fd6a8eb01e9fdfb8f0336585bfd1e2317f2b7f9b37fafe0edebc1, -)
    activity(programs:gunzip, 2021-01-14T14:56:54, 2021-01-14T14:56:54)
    wasAssociatedWith(programs:gunzip,
envs:8289108a0e9fd6a8eb01e9fdfb8f0336585bfd1e2317f2b7f9b37fafe0edebc1, -)
    activity(programs:sed_gz, 2021-01-14T14:56:54, 2021-01-14T14:56:54)
    wasAssociatedWith(programs:sed_gz,
envs:8289108a0e9fd6a8eb01e9fdfb8f0336585bfd1e2317f2b7f9b37fafe0edebc1, -)
    activity(programs:sed_column, 2021-01-14T14:56:54, 2021-01-14T14:56:54)
    wasAssociatedWith(programs:sed_column,
envs:8289108a0e9fd6a8eb01e9fdfb8f0336585bfd1e2317f2b7f9b37fafe0edebc1, -)
```

```
    activity(programs:fastani, 2021-01-14T15:10:57, 2021-01-14T15:13:43)
    wasAssociatedWith(programs:fastani,
envs:8289108a0e9fd6a8eb01e9fdfb8f0336585bfd1e2317f2b7f9b37fafe0edebc1, -)
    used(files:fastani_input, programs:fastani, -)
    wasGeneratedBy(files:fastani_output, programs:fastani, -)
    activity(programs:format_fastani_output, 2021-01-14T15:13:43,
2021-01-14T15:13:44)
    wasAssociatedWith(programs:format_fastani_output,
envs:8289108a0e9fd6a8eb01e9fdfb8f0336585bfd1e2317f2b7f9b37fafe0edebc1, -)
    used(files:fastani_output, programs:format_fastani_output, -)
    wasGeneratedBy(files:fastani_output_fmt, programs:format_fastani_-
output, -)
    activity(programs:cluster_and_plot, 2021-01-14T15:13:44,
2021-01-14T15:13:46)
    wasAssociatedWith(programs:cluster_and_plot,
envs:8289108a0e9fd6a8eb01e9fdfb8f0336585bfd1e2317f2b7f9b37fafe0edebc1, -)
    used(files:fastani_output_fmt, programs:cluster_and_plot, -)
    wasGeneratedBy(files:dendrogram, programs:cluster_and_plot, -)
  endBundle
  bundle samples:GCA_900473955.1
    default <GCA_900473955.1>
    prefix samples <Samples associated with bioprov Project 'genomes_10'>
    prefix project <Project 'genomes_10' with 10 samples>
    prefix files <Files associated with Sample GCA_900473955.1>
    prefix programs <Programs associated with Sample GCA_900473955.1>
    prefix envs <Environments associated with User 'vini'>

    entity(samples:GCA_900473955.1)
    wasDerivedFrom(samples:GCA_900473955.1, project:Project 'genomes_10'␣
 ↪with 10
samples, -, -, -)
    entity(files:genome_assembly)
    wasDerivedFrom(files:genome_assembly, samples:GCA_900473955.1, -, -, -)
    entity(files:proteins)
    wasDerivedFrom(files:proteins, samples:GCA_900473955.1, -, -, -)
    entity(files:genes)
    wasDerivedFrom(files:genes, samples:GCA_900473955.1, -, -, -)
    activity(programs:prodigal, 2021-01-14T15:08:38, 2021-01-14T15:08:55)
    wasAssociatedWith(programs:prodigal,
envs:8289108a0e9fd6a8eb01e9fdfb8f0336585bfd1e2317f2b7f9b37fafe0edebc1, -)
    used(files:genome_assembly, programs:prodigal, -)
```

```
    wasGeneratedBy(files:proteins, programs:prodigal, -)
    wasGeneratedBy(files:genes, programs:prodigal, -)
  endBundle


 [...]
```

Para exportar o documento de proveniência no formato PROV-N ou no formato gráfico, pode-se usar os métodos `prov.write_provn()` ou `prov.dot.write_pdf()`:

[13]:
```
prov.write_provn("genomes_10_provn.txt")  # Escreve no formato PROV-N
prov.dot.write_png("genomes_10.png")  # Escreve em formato gráfico
```

Para fins demonstrativos, vamos analisar o formato gráfico se fosse exportado com uma única amostra. Também vamos remover alguns programas e arquivos para tornar a visualização mais clara.

[14]:
```
# Remover amostras
proj = bp.load_project("genomes_10")
proj.tag = "genomes_1"
proj.samples = {k: v for k, v in proj.items() if k == 'GCA_900473955.1'}

# Remover alguns programas e arquivos
proj.programs = {k: v for k, v in proj.programs.items() if k in
                 ('fastani', 'format_fastani_output', 'cluster_and_plot')
                }
proj.files = {k: v for k, v in proj.files.items() if k in
              ('fastani_input', 'fastani_output', 'fastani_output_fmt',␣
 ↪'dendrogram')
             }

# Criar e exportar o documento de proveniência
prov = bp.BioProvDocument(proj)
prov.dot.write_png("genomes_1.png")
```

A Figura 4.2 resultante representa um grafo para um *workflow* hipotético com duas amostras.

Assim como o documento, a figura é dividida entre três *bundles*: o *bundle* do projeto (no centro), o *bundle* da amostra (à direita) e o *bundle* dos usuários (acima). Os dois primeiros vão conter **entidades**, identificadas pelos ícones ovais amarelos, e **atividades**, identificados por ícones retangulares azuis. O último *bundle* vai conter **agentes** de prove-

niência, identificados pelos pentágonos laranjas. Entidades contemplam amostras, arquivos e o projeto em si. Atividades contemplam os programas utilizados para processar os arquivos, e agentes descrevem o(s) ambiente(s) computacional e usuário(s) associado(s).

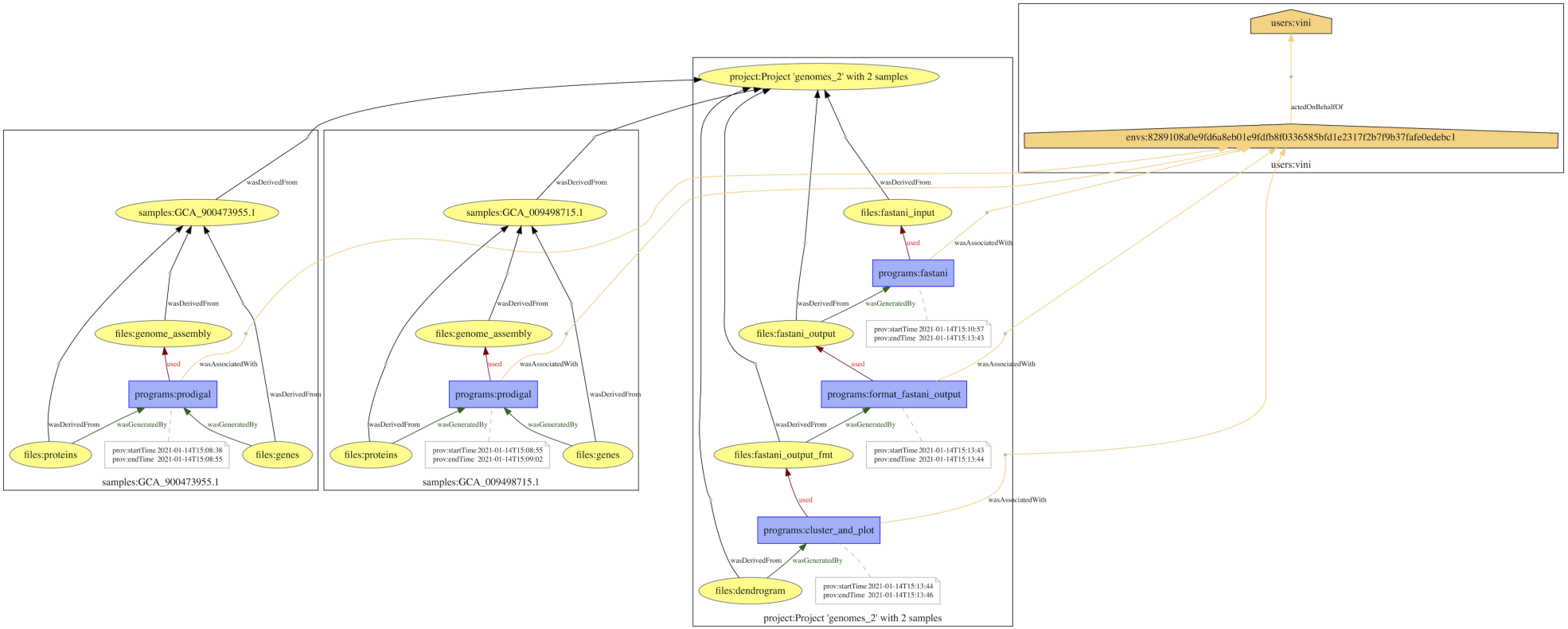


Figura 4.2: Grafo de proveniência criado a partir do objeto `BioProvDocument`. As setas entre os ícones representam relacionamentos de proveniência.

São identificados cinco diferentes tipos de relacionamentos de proveniência na Figura 4.2:

Tabela 4.4: Relacionamentos de proveniência em um documento BioProv, de acordo com a recomendação W3C-PROV.

Nome da relação	**Elementos envolvidos**
actedOnBehalfOf	Agent ⇒ Agent
used	Activity ⇒ Entity
wasAssociatedWith	Activity ⇒ Agent
wasGeneratedBy	Entity ⇒ Activity
wasDerivedFrom	Entity ⇒ Entity

Amostras e arquivos de projeto são entidades derivadas de um projeto. Arquivos de amostras específicas, por sua vez, são derivadas da entidade da amostra. Entidades de arquivos também possuem os relacionamentos "used" e "wasGeneratedBy" em relação a atividades, se serviram como arquivos de entrada ou saída para os programas correspondentes as atividades. As atividades também são associadas a um ambiente computacional, que por sua vez, age em nome de um usuário. É possível ter múltiplos usuários e ambientes, porém cada atividade só poderá estar associada a um ambiente.

Tais relacionamento de proveniência ajudam a determinar que arquivo foi gerado por qual programa, quem foi o usuário responsável por esse programa, e também as derivações entre amostras e arquivos.

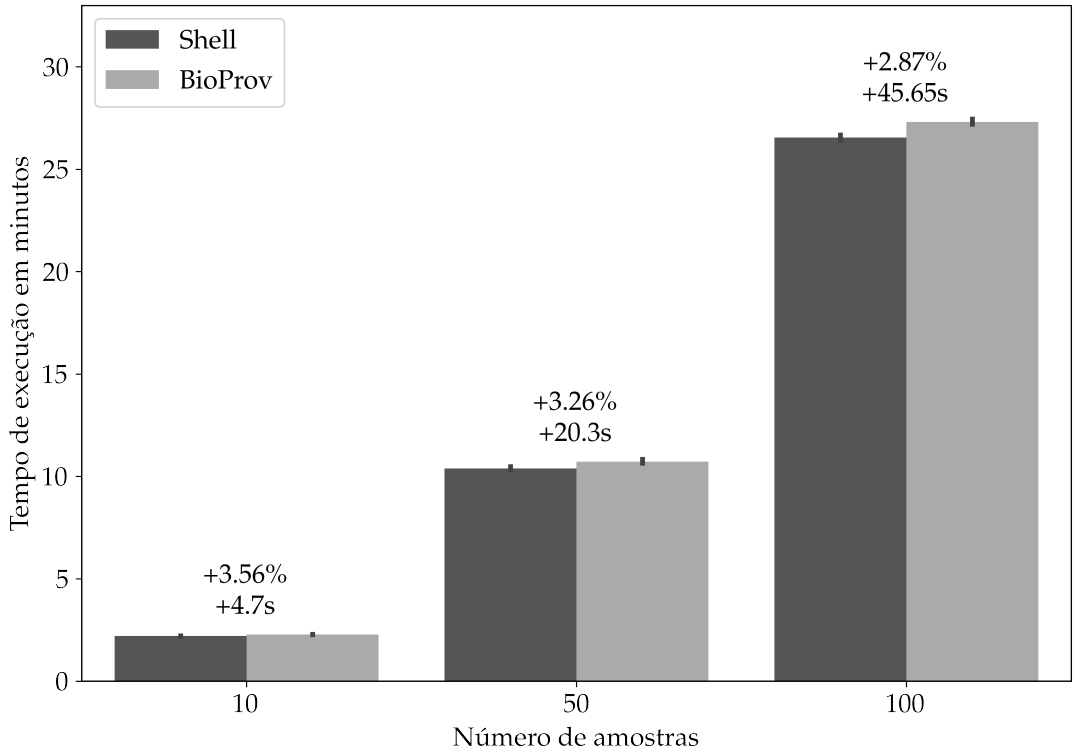
4.2.4 Carga computacional

Para avaliar a carga computacional ocasionada pela coleta de dados com BioProv, o *workflow* de taxonomia genômica foi executado cinco vezes com três conjuntos de dados: 10, 50 e 100 amostras. O *workflow* foi executado com o script BioProv e com um script Shell contendo somente os comandos dos programas executados. O tempo foi medido em segundos para cada execução com o comando `time` do UNIX[10]. A etapa de pré-processamento foi omitida, pois faz o download de dados e depende da velocidade de rede.

A Figura 4.3 mostra o tempo de execução em minutos para BioProv e a execução com Shell, direto no interpretador do sistema. A sobrecarga foi desprezível, variando entre 2.87 - 3.56%. A sobrecarga relativa foi inversamente proporcional ao número de amostras em cada execução: a maior sobrecarga relativa (de 3.56%) foi para a execução no conjunto de dados de 10 amostras. A tabela com os valores de todos os tempos de execução encontra-se no Apêndice (Tabela A.1).

[10] https://man.cx/time

Figura 4.3: Tempo de execução ($N = 5$) do *workflow* de taxonomia genômica executado com BioProv (em cinza) e diretamente na linha de comando do sistema ("Shell", em preto). Acima de cada par de barras está a diferença média de tempo absoluta (em segundos) e relativa.



4.3 Sumário

Foi realizado um estudo de caso de uso da biblioteca BioProv para um *workflow* de taxonomia genômica. Uma versão modificada desse *workflow* foi usado para determinar o status taxonômico de um grupo de picocianobactérias, que foi publicado em dois artigos ao longo da construção da presente dissertação. Para fins de avaliação do desempenho da BioProv, o experimento foi realizado em um servidor de 40 processadores da UFRJ. Foi possível instalar a biblioteca com um único comando no ambiente desejado, instrumentar o *workflow* em um script Python, executar o *workflow* com diferentes conjuntos de dados e armazenar os dados de proveniência no banco de dados interno da biblioteca. A sobrecarga computacional (medida como tempo adicional de execução em segundos) foi de no mínimo 2.87% e no máximo 3.56%. Os dados capturados podem ser carregados em um interpretador interativo como o ambiente Jupyter e consultados com sintaxe Python, e exportados como um documento de proveniência no formato W3C-PROV. Essa demonstração ilustra a eficácia e as limitações da biblioteca, bem como funcionalidades importantes para usuários que desejam colocar a biblioteca em produção.

Capítulo 5

Conclusões

É notável que a bioinformática é uma ciência guiada por dados, e que nos últimos anos sofreu um aumento vertiginoso no volume de dados disponíveis publicamente. Isso correspondeu a uma limitação na capacidade de análise de dados, e em desafios relacionados à reprodutibilidade e validade dessas análises. a recomendação W3C-PROV foi criada com o propósito de ser um formato interoperável para representação de proveniência, e a popularização de sistemas de gerência de *workflow* científico impulsionou a capacidade de orquestração e execução de *workflows* complexos. Essa dissertação apresenta uma solução para a extração de proveniência em *workflows* de bioinformática. A necessidade desse esforço é justificada pela ausência de soluções de captura de proveniência que sejam específicas para o domínio da bioinformática, e adicionalmente que suportem um formato estruturado e interoperável, como o W3C-PROV.

Essa dissertação apresenta uma arquitetura com componentes para extração e armazenamento de dados de proveniência, e que propõe uma representação de classes de alto nível para o domínio da bioinformática. Essa representação contempla elementos como amostras, arquivos, programas e usuários e é integrada ao modelo de dados W3C-PROV. A arquitetura é disponibilizada por meio de uma biblioteca de componentes a serem invocados por meio da instrumentação do *workflow* em scripts Python ou através da aplicação de linha de comando. A implementação é caracterizada em detalhes, com descrições de cada classe e componente, para desenvolvedores(as) interessados(as) possam compreender e/ou modificar o código fonte, e está disponível no repositório do GitHub, junto de tutoriais e dados para teste. Adicionalmente, a biblioteca é distribuída pelo PyPI e pelo canal Bioconda, que agrega pacotes de bioinformática.

Para avaliar a funcionalidade de BioProv, foi realizado um experimento envolvendo execuções de um *workflow* de taxonomia genômica em três conjuntos de dados de tamanhos diferentes, e que serve como demonstração das capacidades da solução. Os dados de proveniência do *workflow* foram capturados de forma automática com uma sobrecarga computacional desprezível, com um acréscimo consistente de menos de 5% de tempo de

execução. Foi possível realizar consultas analíticas desses através de um interpretador interativo do ambiente Jupyter, obtendo informações como tempo de execução e dados de domínio dos arquivos gerados pelo workflow.

Em sumário, as conclusões dessa dissertação podem ser resumidas pelos seguintes pontos:

- a captura de proveniência é fundamental para a prática de pesquisa em bioinformática, no entanto existem dificuldades na sua implementação;
- não existem bibliotecas para captura de proveniência que sejam especializadas em aplicações de bioinformática. Similarmente, sistemas de gerência de *workflows* científicos populares na bioinformática não oferecem suporte à recomendação W3C-PROV;
- foi desenvolvida uma proposta de solução para esse problema através da implementação de um pacote Python para a criação de documentos de proveniência descrevendo *workflows* de bioinformática;
- embora essa solução ainda esteja em um estágio precoce, se apresenta como uma contribuição sólida de *software* e que pode trazer grandes benefícios para a comunidade.

Direções futuras

Pacotes científicos, em sua maioria, são desenvolvidos por grupos de pesquisa pequenos, restritos a uma até meia-dúzia de pessoas, ou, no caso de pacotes maiores e mais consolidados, são gerenciados por uma comunidade na qual os esforços de desenvolvimento são descentralizados e contribuições externas são regularmente submetidas e avaliadas pelos coordenadores do projeto [117]. No caso de pacotes recém-nascidos, como BioProv, todas as tarefas de desenvolvimento são acumuladas por um número pequeno de indivíduos, incluindo a criação de novas funcionalidades, manutenção, testes, documentação e divulgação. Portanto, é necessário priorizar certas metas de desenvolvimento que serão de máximo "custo-benefício" em termos de horas de mão-de-obra gastas. Como novas ideias de funcionalidades para BioProv, pode-se destacar as seguintes metas de desenvolvimento:

- refinar relações de proveniência; atualmente, as relações de proveniência suportadas por BioProv são somente as não-qualificadas (vide Tabela 2.1), e não é possível criar relações qualificadas com atributos próprios, o que é suportado pelo modelo W3C-PROV [4]. Idealmente, o usuário deve ser capaz de configurar as relações de proveniência da forma como achar melhor, e ser capaz de editar manualmente as relações em documentos de proveniência já existentes.

- prover suporte para integração com SGWFs; como BioProv não é uma máquina de execução de workflow, e sim uma biblioteca de captura de proveniência, não possui as funcionalidades de uma. Logo, é interessante que seja possível utilizar BioProv em conjunto com um SGWF que possua capacidade para otimizar a execução do workflow, como configuração automática de ambientes virtuais e tolerância a falhas. Já é possível fazer isso, mas seria ideal aprimorar essa funcionalidade através de algum tipo de plugin ou extensão para um determinado SGWF que comunique com a API de BioProv.

Essas funcionalidades aumentariam as capacidades de representação de proveniência no formato W3C-PROV (primeiro ponto) e a sua integração com ferramentas existentes (segundo ponto). A necessidade de adotar um padrão comum para representação de proveniência já foi reconhecida pela comunidade [6]; a recomendação W3C-PROV é ideal para isso, porém é necessário que soluções que o implementem tenham uma gama ampla de funcionalidades para que sejam atrativas para pesquisadores. Mesmo com suas limitações, BioProv representa uma contribuição promissora para o debate da proveniência em *workflows* de bioinformática, ao propor uma representação de classes em alto nível para o domínio da bioinformática e que é compatível com o modelo W3C-PROV.

Referências Bibliográficas

[1] MARKOWETZ, F. "All biology is computational biology", *PLoS Biology*, v. 15, n. 3, pp. 4–7, 2017. ISSN: 15457885. doi: 10.1371/journal.pbio.2002050.

[2] MATTOSO, M., WERNER, C., TRAVASSOS, G. H., et al. "Towards supporting the life cycle of large scale scientific experiments", *International Journal of Business Process Integration and Management*, v. 5, n. 1, pp. 79, 2010. ISSN: 1741-8763. doi: 10.1504/ijbpim.2010.033176.

[3] W3C PROVENANCE WORKING GROUP. *PROV-Overview: An Overview of the PROV Family of Documents*. Relatório técnico, 2013. Disponível em: <https://www.w3.org/TR/prov-overview/>.

[4] GROTH, P., MOREAU, L. *An Introduction to PROV*. Morgan & Claypool, 2013. ISBN: 9781627052214. doi: 10.2200/S00528ED1V01Y201308WEB007.

[5] WILSON, G., BRYAN, J., CRANSTON, K., et al. "Good enough practices in scientific computing", *PLOS Computational Biology*, v. 13, n. 6, pp. e1005510, 2017.

[6] PASQUIER, T., LAU, M. K., TRISOVIC, A., et al. "If these data could talk", *Scientific Data*, v. 4, pp. 1–5, 2017. ISSN: 20524463. doi: 10.1038/sdata.2017.114.

[7] STEPHENS, Z. D., LEE, S. Y., FAGHRI, F., et al. "Big data: Astronomical or genomical?" *PLoS Biology*, v. 13, n. 7, pp. 1–11, 2015. ISSN: 15457885. doi: 10.1371/journal.pbio.1002195.

[8] WOESE, C. R. "A new biology for a new century." *Microbiology and molecular biology reviews : MMBR*, v. 68, n. 2, pp. 173–186, jun 2004. ISSN: 1092-2172 (Print). doi: 10.1128/MMBR.68.2.173-186.2004.

[9] MARDIS, E. R. "A decade's perspective on DNA sequencing technology", *Nature*, v. 470, n. 7333, pp. 198–203, 2011. ISSN: 00280836. doi: 10.1038/nature09796. Disponível em: <http://dx.doi.org/10.1038/nature09796>.

[10] SHERMAN, R. M., SALZBERG, S. L. "Pan-genomics in the human genome era", *Nature Reviews Genetics*, v. 21, n. 4, pp. 243–254, 2020. ISSN: 14710064. doi: 10.1038/s41576-020-0210-7. Disponível em: <http://dx.doi.org/10.1038/s41576-020-0210-7>.

[11] GILBERT, J. A., JANSSON, J. K., KNIGHT, R. "Earth Microbiome Project and Global Systems Biology", *mSystems*, v. 3, n. 3, pp. 1–4, 2018. ISSN: 2379-5077. doi: 10.1128/msystems.00217-17.

[12] CRADDOCK, T., HARWOOD, C. R., HALLINAN, J., et al. "e-Science: relieving bottlenecks in large-scale genome analyses", *Nature reviews microbiology*, v. 6, n. 12, pp. 948, 2008.

[13] SCHOLZ, M. B., LO, C. C., CHAIN, P. S. G. "Next generation sequencing and bioinformatic bottlenecks: The current state of metagenomic data analysis", *Current Opinion in Biotechnology*, v. 23, n. 1, pp. 9–15, 2012. ISSN: 09581669. doi: 10.1016/j.copbio.2011.11.013.

[14] MARX, V. "Biology: The big challenges of big data", *Nature*, v. 498, pp. 255, jun 2013. Disponível em: <https://doi.org/10.1038/498255ahttp://10.0.4.14/498255a>.

[15] PAPAGEORGIOU, L., ELENI, P., RAFTOPOULOU, S., et al. "Genomic big data hitting the storage bottleneck." *EMBnet.journal*, v. 24, 2018. ISSN: 2226-6089. Disponível em: <http://www.ncbi.nlm.nih.gov/pubmed/29782620{%}0Ahttp://www.pubmedcentral.nih.gov/articlerender.fcgi?artid=PMC5958914>.

[16] LUCIANO, J. S., STEVENS, R. D. "E-Science and biological pathway semantics", *BMC Bioinformatics*, v. 8, n. SUPPL. 3, 2007. ISSN: 14712105. doi: 10.1186/1471-2105-8-S3-S3.

[17] LIEW, C. S., ATKINSON, M. P., GALEA, M., et al. "Scientific Workflows: Moving Across Paradigms", *ACM Computing Surveys*, v. 49, n. 4, pp. 1–39, 2017. ISSN: 0360-0300. doi: 10.1145/3012429.

[18] MILES, S., WONG, S. C., FANG, W., et al. "Provenance-based validation of e-science experiments", *Web Semantics*, v. 5, n. 1, pp. 28–38, 2007. ISSN: 15708268. doi: 10.1016/j.websem.2006.11.003.

[19] STEVENS, R., ZHAO, J., GOBLE, C. "Using provenance to manage knowledge of In Silico experiments", *Briefings in Bioinformatics*, v. 8, n. 3, pp. 183–194, 2007. ISSN: 14675463. doi: 10.1093/bib/bbm015.

[20] BUNEMAN, P., KHANNA, S., TAN, W.-C., et al. "Why and Where: A Characterization of Data Provenance", *Lecture Notes in Computer Science*, v. 1973, n. January, pp. 316–330, 2001. Disponível em: <https://repository.upenn.edu/cgi/viewcontent.cgi?referer=https://www.google.com/{&}httpsredir=1{&}article=1209{&}context=cis{_}papers>.

[21] PENG, R. "The Reproducibility Crisis in Science", *Significance*, v. June, pp. 30–32, 2015. ISSN: 17409705. doi: 10.1111/j.1740-9713.2015.00827.x.

[22] NEKRUTENKO, A., TAYLOR, J. "Next-generation sequencing data interpretation: Enhancing reproducibility and accessibility", *Nature Reviews Genetics*, v. 13, n. 9, pp. 667–672, 2012. ISSN: 14710056. doi: 10.1038/nrg3305. Disponível em: <http://dx.doi.org/10.1038/nrg3305>.

[23] EDWARDS, A. "Reproducibility: Team up with industry", *Nature*, v. 531, n. 7594, pp. 299–301, 2016. ISSN: 0028-0836. doi: 10.1038/531299a.

[24] WILKINSON, M. D., DUMONTIER, M., AALBERSBERG, I. J., et al. "The FAIR Guiding Principles for scientific data management and stewardship", *Scientific Data*, v. 3, pp. 1–9, 2016. ISSN: 20524463. doi: 10.1038/sdata.2016.18.

[25] WILSON, G., ARULIAH, D. A., BROWN, C. T., et al. "Best practices for scientific computing", *PLoS Biology*, v. 12, n. 1, pp. e1001745, 2014.

[26] COSTA, F., SILVA, V., DE OLIVEIRA, D., et al. "Capturing and querying workflow runtime provenance with PROV: a practical approach". In: *Proceedings of the Joint EDBT/ICDT 2013 Workshops*, pp. 282–289. ACM, 2013.

[27] YUAN, Z., TON THAT, D., KOTHARI, S., et al. "Utilizing Provenance in Reusable Research Objects", *Informatics*, v. 5, n. 1, pp. 14, 2018. doi: 10.3390/informatics5010014.

[28] YILMAZ, P., KOTTMANN, R., FIELD, D., et al. "Minimum information about a marker gene sequence (MIMARKS) and minimum information about any (x) sequence (MIxS) specifications", *Nature Biotechnology*, v. 29, n. 5, pp. 415–420, 2011. ISSN: 10870156. doi: 10.1038/nbt.1823.

[29] DODSWORTH, J. A., HEDLUND, B., MALMSTROM, R. R., et al. "Minimum information about a single amplified genome (MISAG) and a metagenome-assembled genome (MIMAG) of bacteria and archaea", *Nature Biotechnology*, v. 35, n. 8, pp. 725–731, 2017. ISSN: 1087-0156. doi: 10.1038/nbt.3893.

[30] FIELD, D., GARRITY, G., GRAY, T., et al. "The minimum information about a genome sequence (MIGS) specification", *Nature Biotechnology*, v. 26, pp. 541, may 2008. Disponível em: <https://doi.org/10.1038/nbt1360http://10.0.4.14/nbt1360https://www.nature.com/articles/nbt1360{#}supplementary-information>.

[31] TEN HOOPEN, P., FINN, R. D., BONGO, L. A., et al. "The metagenomic data life-cycle: standards and best practices." *GigaScience*, v. 6, n. 8, pp. 1–11, aug 2017. ISSN: 2047-217X (Electronic). doi: 10.1093/gigascience/gix047.

[32] BROOKSBANK, C., SCHNEIDER, M. V., RADIVOJAC, P., et al. "Bioinformatics Curriculum Guidelines: Toward a Definition of Core Competencies", *PLoS Computational Biology*, v. 10, n. 3, pp. e1003496, 2014. doi: 10.1371/journal.pcbi.1003496.

[33] ATTWOOD, T. K., BLACKFORD, S., BRAZAS, M. D., et al. "A global perspective on evolving bioinformatics and data science training needs", *Briefings in Bioinformatics*, v. 20, n. 2, pp. 398–404, 2017. ISSN: 14774054. doi: 10.1093/bib/bbx100.

[34] PONCE, M., SPENCE, E., VAN ZON, R., et al. "Bridging the Educational Gap between Emerging and Established Scientific Computing Disciplines", *The Journal of Computational Science Education*, v. 10, n. 1, pp. 4–11, 2019. ISSN: 2153-4136. doi: 10.22369/issn.2153-4136/10/1/1.

[35] GONÇALVES, J. C. D. A., DE OLIVEIRA, D., OCAÑA, K. A., et al. "Using domain-specific data to enhance scientific workflow steering queries", *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, v. 7525 LNCS, pp. 152–167, 2012. ISSN: 03029743. doi: 10.1007/978-3-642-34222-6_12.

[36] LESK, A. *Bioinformatics*. Encyclopædia Britannica, 2019.

[37] STEVENS, H. *Life out of sequence: a data-driven history of bioinformatics*. The University of Chicago Press, 2013.

[38] NEEDLEMAN, S. B., WUNSCH, C. D. "A general method applicable to the search for similarities in the amino acid sequence of two proteins", *Journal of Molecular Biology*, v. 48, n. 3, pp. 443–453, mar 1970. ISSN: 00222836. doi: 10.1016/0022-2836(70)90057-4. Disponível em: <https://linkinghub.elsevier.com/retrieve/pii/0022283670900574>.

[39] SMITH, T. F., WATERMAN, M. S. "Identification of common molecular subsequences", *Journal of Molecular Biology*, v. 147, n. 1, pp. 195–197, 1981. ISSN: 00222836. doi: 10.1016/0022-2836(81)90087-5.

[40] BURROWS, M., WHEELER, D. J. "A block-sorting lossless data compression algorithm", 1994.

[41] DURBIN, R., EDDY, S. R., KROGH, A., et al. *Biological Sequence Analysis*. 1998. doi: 10.1017/cbo9780511790492.

[42] ALTSCHUL, S. F., GISH, W., MILLER, W., et al. "Basic local alignment search tool", *Journal of Molecular Biology*, v. 215, n. 3, pp. 403–410, 1990. ISSN: 00222836. doi: 10.1016/S0022-2836(05)80360-2.

[43] LARKIN, M. A., BLACKSHIELDS, G., BROWN, N. P., et al. "Clustal W and Clustal X version 2.0", *bioinformatics*, v. 23, n. 21, pp. 2947–2948, 2007.

[44] ALBERT, I. *The BioStars Handbook 2nd Edition*. 2021. ISBN: 978-0-578-80435-4.

[45] MCMURDIE, P. J., HOLMES, S. "Phyloseq: An R Package for Reproducible Interactive Analysis and Graphics of Microbiome Census Data", *PLoS ONE*, v. 8, n. 4, 2013. ISSN: 19326203. doi: 10.1371/journal.pone.0061217.

[46] GLOOR, G. B., MACKLAIM, J. M., PAWLOWSKY-GLAHN, V., et al. "Microbiome datasets are compositional: And this is not optional". 2017. ISSN: 1664302X.

[47] MCMURDIE, P. J., HOLMES, S. "Waste Not, Want Not: Why Rarefying Microbiome Data Is Inadmissible", *PLoS Computational Biology*, v. 10, n. 4, 2014. ISSN: 15537358. doi: 10.1371/journal.pcbi.1003531.

[48] MCLAREN, M. R., WILLIS, A. D., CALLAHAN, B. J. "Consistent and correctable bias in metagenomic sequencing experiments", *eLife*, v. 8, pp. e46923, sep 2019. ISSN: 2050-084X. doi: 10.7554/eLife.46923. Disponível em: <https://doi.org/10.7554/eLife.46923>.

[49] NELSON, W. C., TULLY, B. J., MOBBERLEY, J. M. "Biases in genome reconstruction from metagenomic data", *PeerJ*, v. 8, pp. e10119, oct 2020. ISSN: 2167-8359. doi: 10.7717/peerj.10119. Disponível em: <https://peerj.com/articles/10119>.

[50] REITER, T., BROOKS, P. T., IRBER, L., et al. "Streamlining data-intensive biology with workflow systems", *GigaScience*, v. 10, n. 1, jan 2021. ISSN: 2047-217X. doi: 10.1093/gigascience/giaa140. Disponível em: <https://academic.oup.com/gigascience/article/doi/10.1093/gigascience/giaa140/6092773>.

[51] BAKER, M. "1,500 scientists lift the lid on reproducibility." may 2016. ISSN: 1476-4687 (Electronic).

[52] KANWAL, S., KHAN, F. Z., LONIE, A., et al. "Investigating reproducibility and tracking provenance - A genomic workflow case study", *BMC Bioinformatics*, v. 18, n. 1, pp. 337, jul 2017. ISSN: 14712105. doi: 10.1186/s12859-017-1747-0. Disponível em: <http://bmcbioinformatics.biomedcentral.com/articles/10.1186/s12859-017-1747-0>.

[53] OCAÑA, K. A. C. S., DE OLIVEIRA, D., SILVA, V., et al. "Exploiting the Parallel Execution of Homology Workflow Alternatives in HPC Compute Clouds". In: *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, pp. 336–350, 2015. doi: 10.1007/978-3-319-22885-3_29. Disponível em: <http://link.springer.com/10.1007/978-3-319-22885-3{_}29>.

[54] OCAÑA, K. A., DE OLIVEIRA, D., DIAS, J., et al. "Discovering drug targets for neglected diseases using a pharmacophylogenomic cloud workflow", *2012 IEEE 8th International Conference on E-Science, e-Science 2012*, 2012. doi: 10.1109/eScience.2012.6404431.

[55] OCAÑA, K. A., DE OLIVEIRA, D., HORTA, F., et al. "Exploring molecular evolution reconstruction using a parallel cloud based scientific workflow", *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, v. 7409 LNBI, pp. 179–191, 2012. ISSN: 03029743. doi: 10.1007/978-3-642-31927-3_16.

[56] OLIVEIRA, D. D., COSTA, F., SILVA, V., et al. "Debugging Scientific Workflows with Provenance: Achievements and Lessons Learned", *Proc. of the SBBD - Brazilian Symp. on Databases*, pp. 67–76, 2014.

[57] LEIPZIG, J. "A review of bioinformatic pipeline frameworks", *Briefings in Bioinformatics*, v. 18, n. 3, pp. 530–536, 2016. ISSN: 1477-4054. doi: 10.1093/bib/bbw020. Disponível em: <https://doi.org/10.1093/bib/bbw020>.

[58] SILVA, V., DE OLIVEIRA, D., VALDURIEZ, P., et al. "DfAnalyzer: Runtime Dataflow Analysis of Scientific Applications using Provenance", v. 11, n. 12, pp. 2082–2085, 2018. doi: 10.14778/3229863.3236265. Disponível em: <https://doi.org/10.14778/3229863.3236265>.

[59] JACKSON, M. J., WALLACE, E., KAVOUSSANAKIS, K. "Using rapid prototyping to choose a bioinformatics workflow management system", *bioRxiv*, p. 2020.08.04.236208, 2020. Disponível em: <https://doi.org/10.1101/2020.08.04.236208>.

[60] DI TOMMASO, P., CHATZOU, M., FLODEN, E. W., et al. "Nextflow enables reproducible computational workflows", *Nature Biotechnology*, v. 35, n. 4, pp. 316–319, 2017. ISSN: 15461696. doi: 10.1038/nbt.3820.

[61] EWELS, P. A., PELTZER, A., FILLINGER, S., et al. "The nf-core framework for community-curated bioinformatics pipelines". feb 2020. ISSN: 15461696.

[62] KÖSTER, J., RAHMANN, S. "Snakemake-a scalable bioinformatics workflow engine", *Bioinformatics*, v. 28, n. 19, pp. 2520–2522, 2012. ISSN: 14602059. doi: 10.1093/bioinformatics/bts480.

[63] HIDALGO, C. G., GUEVARA, M. E., BUCHELI, V. A., et al. "Bioinformatics software for genomic: a systematic review on GitHub", *PeerJ Preprints*, pp. 1–14, 2018. ISSN: 2167-9843. doi: 10.7287/peerj.preprints.27352. Disponível em: <https://doi.org/10.7287/peerj.preprints.27352v3>.

[64] VIVIAN, J., RAO, A. A., NOTHAFT, F. A., et al. "Toil enables reproducible, open source, big biomedical data analyses". 2017. ISSN: 15461696.

[65] AMSTUTZ, P., CRUSOE, M. R., TIJANIĆ, N., et al. "Common Workflow Language, v1.0", 2016. doi: https://doi.org/10.6084/m9.figshare.3115156.v2.

[66] CAMPOS, V. S. *DfA-lib-Python: uma biblioteca para extração de dados científicos usando a DfAnalyzer*. Tese de Doutorado, Universidade Federal do Rio de Janeiro, 2018.

[67] SILVA, V. "DfAnalyzer-Spark". Mar 2018. Disponível em: <https://github.com/vssousa/dfanalyzer-spark>.

[68] PINA, D., NEVES, L., PAES, A., et al. "Análise de Hiperparâmetros em Aplicações de Aprendizado Profundo por meio de Dados de Proveniência". In: *Anais do XXXIV Simpósio Brasileiro de Banco de Dados*, pp. 223–228, Porto Alegre, RS, Brasil, 2019. SBC. doi: 10.5753/sbbd.2019.8827. Disponível em: <https://sol.sbc.org.br/index.php/sbbd/article/view/8827>.

[69] GU, J., WANG, Z., KUEN, J., et al. "Recent advances in convolutional neural networks", *Pattern Recognition*, v. 77, pp. 354–377, 2018. ISSN: 0031-3203. doi: https://doi.org/10.1016/j.patcog.2017.10.013. Disponível em: <http://www.sciencedirect.com/science/article/pii/S0031320317304120>.

[70] PINA, D. B. *Análise de hiperparâmetros no treinamento de redes de aprendizado profundo usando dados de proveniência*. Tese de Mestrado, Universidade Federal do Rio de Janeiro, 2020.

[71] DONG, T. "PROV v2.0.0: A Python library for W3C Provenance Data Model". Nov 2020. Disponível em: <https://github.com/trungdong/prov>.

[72] COCK, P. J. A., ANTAO, T., CHANG, J. T., et al. "Biopython: freely available Python tools for computational molecular biology and bioinformatics", *Bioinformatics*, v. 25, n. 11, pp. 1422–1423, jun 2009. ISSN: 1367-4803. doi: 10.1093/bioinformatics/btp163. Disponível em: <https://academic.oup.com/bioinformatics/article-lookup/doi/10.1093/bioinformatics/btp163>.

[73] HOLLAND, R. C., DOWN, T. A., POCOCK, M., et al. "BioJava: an open-source framework for bioinformatics", *Bioinformatics*, v. 24, n. 18, pp. 2096–2097, 2008.

[74] "BioJulia". 2020. Disponível em: <https://biojulia.net/>.

[75] KÖSTER, J. "Rust-Bio: a fast and safe bioinformatics library", *Bioinformatics*, v. 32, n. 3, pp. 444–446, 2015. ISSN: 1367-4803. doi: 10.1093/bioinformatics/btv573. Disponível em: <https://doi.org/10.1093/bioinformatics/btv573>.

[76] SIEMENS, M. "TinyDB v4.3.0: a lightweight document oriented database". Nov 2020. Disponível em: <https://github.com/msiemens/tinydb>.

[77] RAGAN-KELLEY, M., PEREZ, F., GRANGER, B., et al. "The Jupyter/IPython architecture: a unified view of computational research, from interactive exploration to communication and publication." In: *AGU Fall Meeting Abstracts*, 2014.

[78] MCKINNEY, W. "pandas: a Foundational Python Library for Data Analysis and Statistics", *Python for High Performance and Scientific Computing*, 2011.

[79] ABADI, M., CARDELLI, L. *A theory of objects*. Springer Science & Business Media, 2012.

[80] HYATT, D., CHEN, G.-L., LOCASCIO, P. F., et al. "Prodigal: prokaryotic gene recognition and translation initiation site identification." *BMC bioinformatics*, v. 11, pp. 119, 2010. ISSN: 1471-2105. doi: 10.1186/1471-2105-11-119. Disponível em: <http://www.pubmedcentral.nih.gov/articlerender.fcgi?artid=2848648{&}tool=pmcentrez{&}rendertype=abstract>.

[81] CAMACHO, C., COULOURIS, G., AVAGYAN, V., et al. "BLAST+: architecture and applications." *BMC bioinformatics*, v. 10, pp. 421, dec 2009. ISSN: 1471-2105. doi: 10.1186/1471-2105-10-421. Disponível em: <http://www.ncbi.nlm.nih.gov/pubmed/20003500http://www.pubmedcentral.nih.gov/articlerender.fcgi?artid=PMC2803857>.

[82] BUCHFINK, B., XIE, C., HUSON, D. H. "Fast and sensitive protein alignment using DIAMOND". 2014. ISSN: 15487105.

[83] PRICE, M. N., DEHAL, P. S., ARKIN, A. P. "FastTree 2 - Approximately maximum-likelihood trees for large alignments", *PLoS ONE*, v. 5, n. 3, 2010. ISSN: 19326203. doi: 10.1371/journal.pone.0009490.

[84] MENZEL, P., NG, K. L., KROGH, A. "Fast and sensitive taxonomic classification for metagenomics with Kaiju", *Nature Communications*, v. 7, 2016. ISSN: 20411723. doi: 10.1038/ncomms11257.

[85] BRAY, N. L., PIMENTEL, H., MELSTED, P., et al. "Near-optimal probabilistic RNA-seq quantification", *Nature Biotechnology*, v. 34, n. 5, pp. 525–527, 2016. ISSN: 15461696. doi: 10.1038/nbt.3519.

[86] KATOH, K., KUMA, K. I., TOH, H., et al. "MAFFT version 5: Improvement in accuracy of multiple sequence alignment", *Nucleic Acids Research*, v. 33, n. 2, pp. 511–518, 2005. ISSN: 03051048. doi: 10.1093/nar/gki198.

[87] EDGAR, R. C. "MUSCLE: Multiple sequence alignment with high accuracy and high throughput", *Nucleic Acids Research*, v. 32, n. 5, pp. 1792–1797, 2004. ISSN: 03051048. doi: 10.1093/nar/gkh340.

[88] SEEMANN, T. "Prokka: Rapid prokaryotic genome annotation", *Bioinformatics*, 2014. ISSN: 14602059. doi: 10.1093/bioinformatics/btu153.

[89] LAKIN, S. M., DEAN, C., NOYES, N. R., et al. "MEGARes: An antimicrobial resistance database for high throughput sequencing", *Nucleic Acids Research*, v. 45, n. D1, pp. D574–D580, jan 2017. ISSN: 13624962. doi: 10.1093/nar/gkw1009.

[90] PARKS, D. H., RINKE, C., CHUVOCHINA, M., et al. "Recovery of nearly 8,000 metagenome-assembled genomes substantially expands the tree of life", *Nature Microbiology*, v. 2, n. 11, pp. 1533–1542, 2017. ISSN: 20585276. doi: 10.1038/s41564-017-0012-7. Disponível em: <http://dx.doi.org/10.1038/s41564-017-0012-7>.

[91] BREITWIESER, F. P., LU, J., SALZBERG, S. L. "A review of methods and databases for metagenomic classification and assembly", *Briefings in Bioinformatics*, v. 20, n. 4, pp. 1125–1139, 2018. ISSN: 14774054. doi: 10.1093/bib/bbx120.

[92] O'LEARY, N. A., WRIGHT, M. W., BRISTER, J. R., et al. "Reference sequence (RefSeq) database at NCBI: Current status, taxonomic expansion, and functional annotation", *Nucleic Acids Research*, 2016. ISSN: 13624962. doi: 10.1093/nar/gkv1189.

[93] KITTS, P. A., CHURCH, D. M., THIBAUD-NISSEN, F., et al. "Assembly: a resource for assembled genomes at NCBI", *Nucleic Acids Research*, v. 44, n. D1, pp. D73–D80, 2015. ISSN: 0305-1048. doi: 10.1093/nar/gkv1226. Disponível em: <https://doi.org/10.1093/nar/gkv1226>.

[94] PARKS, D. H., CHUVOCHINA, M., WAITE, D. W., et al. "A standardized bacterial taxonomy based on genome phylogeny substantially revises the tree of life." *Nature biotechnology*, v. 36, n. 10, pp. 996–1004, 2018. ISSN: 1546-1696. doi: 10.1038/nbt.4229. Disponível em: <http://www.ncbi.nlm.nih.gov/pubmed/30148503>.

[95] CHAUMEIL, P.-A., MUSSIG, A. J., HUGENHOLTZ, P., et al. "GTDB-Tk: a toolkit to classify genomes with the Genome Taxonomy Database", *Bioinformatics*, v. 36, n. 6, pp. 1925–1927, 2019. ISSN: 1367-4803. doi: 10.1093/bioinformatics/btz848. Disponível em: <https://academic.oup.com/bioinformatics/article-abstract/36/6/1925/5626182>.

[96] GEVERS, D., COHAN, F. M., LAWRENCE, J. G., et al. "Reevaluating prokaryotic species", *Nature Reviews Microbiology*, v. 3, n. September, pp. 733–739, 2005.

[97] RODRIGUEZ-R, L. M., KONSTANTINIDIS, K. T. "Bypassing Cultivation To Identify Bacterial Species", *Microbe*, v. 9, n. 3, pp. 111–117, 2014. Disponível em: <http://enve-omics.gatech.edu/sites/default/files/2014-Rodriguez{_}R-Konstantinidis{_}Microbe{_}Magazine.pdf>.

[98] THOMPSON, C. C., CHIMETTO, L., EDWARDS, R. A., et al. "Microbial genomic taxonomy", *BMC Genomics*, v. 14, n. 1, 2013. ISSN: 14712164. doi: 10.1186/1471-2164-14-913.

[99] HUGENHOLTZ, P., SKARSHEWSKI, A., PARKS, D. H. "Genome-based microbial taxonomy coming of age", *Cold Spring Harbor Perspectives in Biology*, v. 8, n. 6, pp. a018085, 2016.

[100] KONSTANTINIDIS, K. T., TIEDJE, J. M. "Towards a genome-based taxonomy for prokaryotes", *Journal of Bacteriology*, v. 187, n. 18, pp. 6258–6264, 2005. ISSN: 00219193. doi: 10.1128/JB.187.18.6258-6264.2005.

[101] COLEMAN, M. L., CHISHOLM, S. W. "Code and context: Prochlorococcus as a model for cross-scale biology", *Trends in Microbiology*, v. 15, n. 9, pp. 398–407, 2007. ISSN: 0966842X. doi: 10.1016/j.tim.2007.07.001.

[102] ABED, R., DOBRETSOV, S., SUDESH, K. "Applications of cyanobacteria in biotechnology", *Journal of Applied Microbiology*, v. 106, n. 1, pp. 1–12, jan 2009. ISSN: 13645072. doi: 10.1111/j.1365-2672.2008.03918.x. Disponível em: <http://doi.wiley.com/10.1111/j.1365-2672.2008.03918.x>.

[103] BILLER, S. J., BERUBE, P. M., LINDELL, D., et al. "Prochlorococcus: The structure and function of collective diversity", *Nature Reviews Microbiology*, v. 13, n. 1, pp. 13–27, 2015. ISSN: 17401534. doi: 10.1038/nrmicro3378. Disponível em: <http://dx.doi.org/10.1038/nrmicro3378>.

[104] FLOMBAUM, P., GALLEGOS, J. L., GORDILLO, R. A., et al. "Present and future global distributions of the marine Cyanobacteria Prochlorococcus and Synechococcus", *Proceedings of the National Academy of Sciences*, v. 110, n. 24, pp. 9824–9829, 2013. ISSN: 0027-8424. doi: 10.1073/pnas.1307701110. Disponível em: <http://www.pnas.org/cgi/doi/10.1073/pnas.1307701110>.

[105] COUTINHO, F., TSCHOEKE, D. A., THOMPSON, F., et al. "Comparative genomics of Synechococcus and proposal of the new genus Parasynechococcus", *PeerJ*, v. 4, pp. e1522, 2016. doi: 10.7717/peerj.1522.

[106] COUTINHO, F. H., DUTILH, B. E., THOMPSON, C. C., et al. "Proposal of fifteen new species of Parasynechococcus based on genomic, physiological and ecological features", *Archives of Microbiology*, v. 198, n. 10, pp. 973–986, 2016. ISSN: 1432072X. doi: 10.1007/s00203-016-1256-y.

[107] THOMPSON, C. C., SILVA, G. G., VIEIRA, N. M., et al. "Genomic Taxonomy of the Genus Prochlorococcus", *Microbial Ecology*, v. 66, n. 4, pp. 752–762, 2013. ISSN: 00953628. doi: 10.1007/s00248-013-0270-8.

[108] WALTER, J. M., COUTINHO, F. H., DUTILH, B. E., et al. "Ecogenomics and taxonomy of Cyanobacteria phylum", *Frontiers in Microbiology*, v. 8, n. NOV, 2017. ISSN: 1664302X. doi: 10.3389/fmicb.2017.02132.

[109] BERUBE, P. M., BILLER, S. J., HACKL, T., et al. "Data descriptor: Single cell genomes of Prochlorococcus, Synechococcus, and sympatric microbes from diverse marine environments", *Scientific Data*, v. 5, n. March, pp. 1–11, 2018. ISSN: 20524463. doi: 10.1038/sdata.2018.154.

[110] TSCHOEKE, D. A., SALAZAR, V., VIDAL, L. M., et al. "Unlocking the genomic taxonomy of the Prochlorococcus collective", *Microbial Ecology*, pp. 1–13, mar 2020. doi: 10.1007/s00248-020-01526-5. Disponível em: <https://doi.org/10.1007/s00248-020-01526-5>.

[111] SALAZAR, V. W., TSCHOEKE, D. A., SWINGS, J., et al. "A new genomic taxonomy system for the Synechococcus collective", *Environmental Microbiology*, pp. 1462–2920.15173, aug 2020. ISSN: 1462-2912. doi: 10.1111/1462-2920.15173. Disponível em: <https://sfamjournals.onlinelibrary.wiley.com/doi/abs/10.1111/1462-2920.15173https://onlinelibrary.wiley.com/doi/abs/10.1111/1462-2920.15173>.

[112] ROSSELLÓ-MÓRA, R., TRUJILLO, M. E., SUTCLIFFE, I. C. "Introducing a digital protologue: a timely move towards a database-driven systematics of archaea and bacteria", *Antonie van Leeuwenhoek, International Journal of General and Molecular Microbiology*, v. 110, n. 4, pp. 455–456, 2017. ISSN: 15729699. doi: 10.1007/s10482-017-0841-7.

[113] STACKEBRANDT, E., SMITH, D. "Paradigm shift in species description: the need to move towards a tabular format", *Archives of Microbiology*, v. 201, n. 2, pp. 143–145, 2019. ISSN: 1432072X. doi: 10.1007/s00203-018-1609-9. Disponível em: <http://dx.doi.org/10.1007/s00203-018-1609-9>.

[114] SAYERS, E. W., CAVANAUGH, M., CLARK, K., et al. "GenBank", *Nucleic Acids Research*, v. 48, n. D1, pp. D84–D86, 2020. ISSN: 13624962. doi: 10.1093/nar/gkz956.

[115] JAIN, C., RODRIGUEZ-R, L. M., PHILLIPPY, A. M., et al. "High throughput ANI analysis of 90K prokaryotic genomes reveals clear species boundaries", *Nature communications*, v. 9, n. 1, pp. 5114, nov 2018. ISSN: 2041-1723. doi: 10.1038/s41467-018-07641-9. Disponível em: <https://www.ncbi.nlm.nih.gov/pubmed/30504855https://www.ncbi.nlm.nih.gov/pmc/articles/PMC6269478/>.

[116] VOORHEES, E. M. "Implementing agglomerative hierarchic clustering algorithms for use in document retrieval", *Information Processing & Management*, v. 22, n. 6, pp. 465–476, 1986.

[117] NOWOGRODZKI, A. "How to support open-source software and stay sane", *Nature*, v. 571, n. 7763, pp. 133–135, 2019.

Apêndice A

Material suplementar

A seguir está a tabela de dados completa usada no Capítulo 4 para gerar a Figura 4.3.

Tabela A.1: Tempos de execução para o workflow de taxonomia genômica.

Número de amostras	Método de execução	Tempo de execução em segundos
10	BioProv	137.524
10	BioProv	136.083
10	BioProv	136.498
10	BioProv	136.657
10	BioProv	137.206
10	Shell	130.887
10	Shell	132.372
10	Shell	133.051
10	Shell	131.581
10	Shell	132.57
50	BioProv	641.582
50	BioProv	637.012
50	BioProv	641.835
50	BioProv	656.036
50	BioProv	639.419
50	Shell	621.663
50	Shell	634.02
50	Shell	619.701
50	Shell	620.332
50	Shell	618.66
100	BioProv	1631.593
100	BioProv	1634.887
100	BioProv	1631.357
100	BioProv	1637.181
100	BioProv	1656.117
100	Shell	1580.071
100	Shell	1590.444
100	Shell	1604.902
100	Shell	1588.647
100	Shell	1598.814

Apêndice B

Tutoriais

Além da página de documentação, BioProv conta com dois tutoriais, um introdutório, apresentando a biblioteca, e um apresentando os documentos de proveniência. Os tutoriais estão disponíveis junto do código fonte no formato Jupyter Notebook[1], e são reproduzidos na íntegra a seguir. É importante notar que os tutoriais estão em inglês (assim como a página da documentação) para incluir um público-alvo maior.

B.1 Introduction to BioProv

BioProv is a library to record provenance information of bioinformatics workflows. If you work with genomics, you've probably encountered the situation where you have several different files for a number of biological samples, and each file concerns a certain aspect of your data. As you develop your analysis workflow, it is challenging to keep track of the **provenance** of your data: how, when and why each file was created and/or modified. There are many tools to aid in this task, such as version control, scientific workflow management systems, or even simply keeping a tidy computational notebook.

Although these practices are certainly helpful and we recommend that you employ them, it is not trivial to integrate and share provenance information across different people, research groups and even computing environments. A solution to this has been the development of W3C-PROV, a standard created by the W3C organization to facilitate the exchange of provenance data in the web.

The W3C-PROV is composed of a set of 12 documents, of which maybe the most pertinent to us is the W3C-PROV-DM, which describes a data model to represent provenance information. Although this model is widely implemented in a range of domain applications, including to scientific workflows, to the best of our knowledge, there is not yet a software tool specialized in the provenance of biological data structures and bi-

[1]Que podem ser executados no navegador através da plataforma Binder, como explicado na Seção 3.2.

oinformatics workflows. To extract provenance attributes of common file formats and common project organization patterns in bioinformatics, generic provenance extraction systems must be extended or customized, which can be a costly task for both the domain specialist and the developers of said systems. In order to fulfill this gap, we present BioProv, which aims to facilitate the provenance extraction in bioinformatics workflows by providing a Python library which integrates two open source libraries: BioPython and Prov.

How it works

BioProv is **project-based**, where each **Project** contains a number of **Samples** which have associated **Files**. **Files** may also be associated directly with the **Project**, if they contain information about zero or multiple samples. BioProv also stores information about **Programs** used create new and modify existing **Files**. **Programs** may contain **Parameters** which will determine how they will be run. Once a **Program** has been run, information about the process will be stored as a **Run**.

Therefore, these are the main classes of the BioProv library:

- Project
- Sample
- Files
- Programs
- Parameters
- Run

See an example on how to make a BioProv Project.

A **Project** is composed of **Samples**. We are going to create a **Project** with only 1 **Sample**, which is going to be the genome sequence of *Synechococcus elongatus* PCC 6301, a cyanobacteria. We are going to add attributes to this **Sample** using the *attributes* argument, which takes a Python `dict`. We are going to fill it with information about its location on NCBI.

We then create a **Project** using a list of **Samples**.

[1]:
```
import bioprov as bp

sample = bp.Sample("Synechococcus_elongatus_PCC_6301",
                   attributes={"ncbi_accession": "GCF_000010065.1",
                               "ncbi_database": "assembly"}
                  )
```

```
project = bp.Project(samples=[sample,], tag="introduction")
```

Adding Files and Programs

Now we have a **Project** containing 1 **Sample**. However, our sample has no associated **Files** nor **Programs**. Let's add a **File** to our **Sample** and run a program on it.

BioProv comes with an auxiliary `data` subpackage, which contains some preset data for us to experiment with. The `synechococcus_genome` variable is an instance of `pathlib.PosixPath`, which is used to hold file paths.

[2]:
```
from bioprov.data import synechococcus_genome

# We create a File object based on a path or a string representing a path.
assembly_file = bp.File(synechococcus_genome, tag="assembly")

# We can add this File to our Sample
sample.add_files(assembly_file)
sample.files
```

[2]:
```
{'assembly': /Users/vini/anaconda3/envs/bioprov/lib/python3.7/site-
packages/bioprov/data/genomes/GCF_000010065.1_ASM1006v1_genomic.fna}
```

Now our instance of `Sample` holds a `File` object. Files can be accessed by the attribute `.files`, which is a dictionary composed of `{file.tag: File instance}`.

We can now run a **Program** in our **Sample**. The sample's **Files** can be used as **Parameter** to the program. Programs are processed by the UNIX shell.

Here we are setting up a program using UNIX's `grep` to count the occurrences of a particular kmer in our sample. We are then going to write the results to a new **File**.

To write our program, we start with an instance of the **Program** class and add **Parameters** to it.

[3]:
```
grep = bp.Program("grep")
```

Now that we have a **Program**, let's set the other variables we'll need.

The first is an easy one, the kmer we are going to count:

[4]:
```
kmer = "GATTACA"
```

Then, we need to create a new **File** in our **Sample**. The files associated with a **Sample** can be accessed in the `Sample.files` dictionary:

[5]:
```
sample.files
```

[5]:
```
{'assembly': /Users/vini/anaconda3/envs/bioprov/lib/python3.7/site-
packages/bioprov/data/genomes/GCF_000010065.1_ASM1006v1_genomic.fna}
```

We are going to create a new **File** in this dictionary. To set the path of this **File**, we can use attributes from the existing files.

Each item in the `Sample.files` dictionary is an instance of `bioprov.File`, so there are several attributes which may be useful:

[6]:
```
print(sample.files['assembly'].__class__, "\n")
sample.files['assembly'].__dict__
```

```
<class 'bioprov.src.files.File'>
```

[6]:
```
{'path': PosixPath('/Users/vini/anaconda3/envs/bioprov/lib/python3.7/site-
packages/bioprov/data/genomes/GCF_000010065.1_ASM1006v1_genomic.fna'),
 'name': 'GCF_000010065.1_ASM1006v1_genomic',
 'basename': 'GCF_000010065.1_ASM1006v1_genomic.fna',
 'directory': PosixPath('/Users/vini/anaconda3/envs/bioprov/lib/python3.7/
 ↪site-
packages/bioprov/data/genomes'),
 'extension': '.fna',
 'tag': 'assembly',
 'attributes': {},
 '_exists': True,
 '_size': '2.6 MB',
 'raw_size': 2730026,
 '_sha1': 'f8658496b343257690f828ec14226644dc9e9ca2',
 '_entity': None}
```

For example, if we want to create the new file in the same directory as the `'assembly'` file, we can use its `File.directory` attribute:

[7]:
```
newfilepath = sample.files['assembly'].directory.joinpath(f"{kmer}_count.
 ↪txt")
```

We can now set a new file in the `Sample.files` dictionary based on our kmer and the directory of the existing `'assembly'` **File**:

[8]:
```
sample.files[f'{kmer}_count'] = bp.File(newfilepath)


# which is the same as:
```

```
sample.add_files(bp.File(newfilepath, tag=f"{kmer}_count"))
```

```
Updating file GATTACA_count with value
/Users/vini/anaconda3/envs/bioprov/lib/python3.7/site-
packages/bioprov/data/genomes/GATTACA_count.txt.
```

Running Programs

We now have a new **File** with the name of our kmer in the `Sample.files` dictionary.

Now, we must create the parameters to be added to the `grep` program. Parameters are strings which are added to the program's command-line. We can just put a string with all of our parameters, but creating them one by one and enclosing them with the `bp.Parameter` class will allow for querying later. Parameters are added to a **Program** with the `Program.add_parameter()` method. We then bind the **Program** to the **Sample** using the `Sample.add_programs()` method. It's important to remember these two methods: `Program.add_parameter()` and `Sample.add_programs()`. They allow BioProv to resolve internal relationships between each class.

Finally, we check our command is correct: each `bioprov.Program` instance has a `Program.cmd` attribute which shows the exact command-line which will be run on the UNIX shell.

[9]:
```
count = bp.Parameter("-c")
kmer_param = bp.Parameter(f"'{kmer}'")
in_file = bp.Parameter(str(sample.files['assembly']))
pipe_out = bp.Parameter(">", str(sample.files[f'{kmer}_count']))

for param in (count, kmer_param, in_file, pipe_out):
    grep.add_parameter(param)

sample.add_programs(grep)

grep.cmd
```

[9]:
```
"/usr/bin/grep -c 'GATTACA'
/Users/vini/anaconda3/envs/bioprov/lib/python3.7/site-
packages/bioprov/data/genomes/GCF_000010065.1_ASM1006v1_genomic.fna >
/Users/vini/anaconda3/envs/bioprov/lib/python3.7/site-
packages/bioprov/data/genomes/GATTACA_count.txt"
```

Now we want to run our program. We use the `Program.run()` method.

[10]:
```
grep.run()
```

```
Running program 'grep'.
Command is:
/usr/bin/grep \
        -c 'GATTACA' \
        /Users/vini/anaconda3/envs/bioprov/lib/python3.7/site-
packages/bioprov/data/genomes/GCF_000010065.1_ASM1006v1_genomic.fna > \
        /Users/vini/anaconda3/envs/bioprov/lib/python3.7/site-
packages/bioprov/data/genomes/GATTACA_count.txt
```

```
[10]: Run of Program 'grep' with 4 parameter(s).
      Started at Tue Nov 24 12:13:37 2020.
      Ended at Tue Nov 24 12:13:38 2020.
      Status is finished.
```

When we run a **Program**, we create a new **Run**. The `bioprov.Run` class holds information about a process, such as the start time and end time. Runs are stored in the `Program.runs` attribute:

```
[11]: grep.runs
```

```
[11]: {'1': Run of Program 'grep' with 4 parameter(s).
       Started at Tue Nov 24 12:13:37 2020.
       Ended at Tue Nov 24 12:13:38 2020.
       Status is finished.}
```

```
[12]: # Each Run has useful attributes such as stdout, stderr and status
      grep.runs['1'].__dict__
```

```
[12]: {'program': Program 'grep' with 4 parameter(s).,
       'cmd': "/usr/bin/grep -c 'GATTACA'
      /Users/vini/anaconda3/envs/bioprov/lib/python3.7/site-
      packages/bioprov/data/genomes/GCF_000010065.1_ASM1006v1_genomic.fna >
      /Users/vini/anaconda3/envs/bioprov/lib/python3.7/site-
      packages/bioprov/data/genomes/GATTACA_count.txt",
       'params': OrderedDict([('-c', Parameter with command string '-c '),
                    ("'GATTACA'", Parameter with command string ''GATTACA' '),
                    ('/Users/vini/anaconda3/envs/bioprov/lib/python3.7/site-
      packages/bioprov/data/genomes/GCF_000010065.1_ASM1006v1_genomic.fna',
                     Parameter with command string
      '/Users/vini/anaconda3/envs/bioprov/lib/python3.7/site-
      packages/bioprov/data/genomes/GCF_000010065.1_ASM1006v1_genomic.fna '),
                    ('>',
                     Parameter with command string '>
```

```
/Users/vini/anaconda3/envs/bioprov/lib/python3.7/site-
packages/bioprov/data/genomes/GATTACA_count.txt')]),
 'sample': None,
 'process': <subprocess.Popen at 0x7fed3cc5bb90>,
 'stdin': None,
 'stdout': '',
 'stderr': '',
 '_auto_suppress_stdout': True,
 'start_time': 'Tue Nov 24 12:13:37 2020',
 'end_time': 'Tue Nov 24 12:13:38 2020',
 'duration': '0:00:00.138345',
 'started': True,
 'finished': True,
 '_status': 'Finished',
 'user': 'vini',
 'env': 'a8551940c37e67fe37598975e09e91e084f5da78'}
```

Exporting Projects

We now have a simple, yet complete, **BioProv Project**. We have a **Project** with 1 or more associated **Samples**, and 1 or more **Programs** have been **run** on the sample. We can export this **Project** as a JSON file.

[13]:
```
project['Synechococcus_elongatus_PCC_6301'].files
```

[13]:
```
{'assembly': /Users/vini/anaconda3/envs/bioprov/lib/python3.7/site-
packages/bioprov/data/genomes/GCF_000010065.1_ASM1006v1_genomic.fna,
 'GATTACA_count': /Users/vini/anaconda3/envs/bioprov/lib/python3.7/site-
packages/bioprov/data/genomes/GATTACA_count.txt}
```

[14]:
```
project.to_json("./introduction.json")
```

```
Created JSON file at ./introduction.json.
```

This project can be easily retrieved with the `bioprov.from_json()` function.

[15]:
```
project = bp.from_json("./introduction.json")
```

This allows us to read and write Projects as JSON files, so we can store and/or query them.

What about the provenance?

We've learned the basics of BioProv, like the main classes, how to create **Projects**, **Samples**, and **Programs.** However, the point of BioProv is to be able to convert these elements to the W3C-PROV format. You can couple BioProv Projects (or any other BioProv object, for that matter) to W3C-PROV elements, allowing them to be exported as W3C-PROV documents, implemented with the Prov library. Continue to the W3C-PROV tutorial.

B.2 W3C-PROV workflows

In the last tutorial we learned about how to start a **Project** in BioProv, along with the main classes of the library. In this tutorial, we are going to look at some additional BioProv functions to facilitate our work. We are also going to produce a W3C-PROV document describing our workflow. Let's dive into it.

Importing data easily

You might be thinking that creating BioProv Projects and Samples one by one is quite repetitive. To facilitate this, it is possible to import data into BioProv using comma- or tab-delimited files, using the BioProv `read_csv()` function.

[1]:
```
import bioprov as bp
from bioprov.data import picocyano_dataset

proj = bp.read_csv(picocyano_dataset, tag="picocyano"); proj
```

[1]:
```
Project 'picocyano' with 5 samples
```

The `picocyano_dataset` variable is simply a path pointing to a comma-delimited file that comes with BioProv:

[2]:
```
picocyano_dataset
```

[2]:
```
PosixPath('/Users/vini/anaconda3/envs/bp-use-case/lib/python3.8/site-
packages/bioprov/data/datasets/picocyano.csv')
```

[3]:
```
# Take a peek at the file
!cat ../../bioprov/data/datasets/picocyano.csv
```

```
sample-id,assembly,taxon
GCF_000010065.1,bioprov/data/genomes/GCF_000010065.1_ASM1006v1_genomic.
↪fna,Synec
hococcus elongatus PCC 6301
```

```
GCF_000007925.1,bioprov/data/genomes/GCF_000007925.1_ASM792v1_genomic.
 ↪fna,Prochl
orococcus marinus CCMP1375
GCA_000316515.1,bioprov/data/genomes/GCF_000316515.1_ASM31651v1_genomic.
 ↪fna,Cyan
obium gracile PCC 6307
GCF_000012625.1,bioprov/data/genomes/GCF_000012625.1_ASM1262v1_genomic.
 ↪fna,Paras
ynechococcus africanus CC9605
GCF_000011485.1,bioprov/data/genomes/GCF_000011485.1_ASM1148v1_genomic.
 ↪fna,Thaum
ococcus swingsii MIT9313
```

You can notice that this is a simple comma-delimited file. If you have a different delimiter, simply pass it to the `sep` argument of `read_csv()`, e.g. if you have a tab-delimited file, type `read_csv(path, sep="\t")`.

BioProv uses Pandas to process delimited files. Because of this, you can also import data from Pandas DataFrames using the `from_df()` function. This is quite handy if you want to process your file for a bit before importing it with BioProv.

[4]:
```
import pandas as pd

df = pd.read_csv(picocyano_dataset)

df['assembly'] = "../../" + df["assembly"]
... # do your processing here

proj = bp.from_df(df, file_cols="assembly", tag="picocyano"); proj
```

[4]:
```
Project 'picocyano' with 5 samples
```

The `from_df()` function has some useful arguments to make sure our data is read correctly.

The first is the `index_col`, which is the column used to read the Sample IDs. This column must contain unique identifiers, it can be passed as an integer (position of the column) or as a string (name of the column). However, we don't have to worry about that because it reads the first column as index_col by default.

The second useful argument is `file_cols`, which is used to specify the columns which contain the path to files in our data. The second column of our dataset, the `"assembly"` column, contains the path to the genome assembly of each sample. This will create an instance of **File** for each **Sample**. The **File** will be tagged with the column

name. The remaining columns will be added as attributes to the **Sample**.

[5]:
```
proj = bp.from_df(df, file_cols=["assembly",], tag="picocyano")

# Creates files tagged with the column name
print(proj.samples, "\n")
print(proj['GCF_000010065.1'].files, "\n")
print(proj['GCF_000010065.1'].attributes)
```

```
{'GCF_000010065.1': Sample 'GCF_000010065.1' with 1 file(s)., 'GCF_-
000007925.1':
Sample 'GCF_000007925.1' with 1 file(s)., 'GCA_000316515.1': Sample
'GCA_000316515.1' with 1 file(s)., 'GCF_000012625.1': Sample 'GCF_-
000012625.1'
with 1 file(s)., 'GCF_000011485.1': Sample 'GCF_000011485.1' with 1␣
 ↪file(s).}

{'assembly': /Users/vini/Bio/BioProv/docs/tutorials/../../bioprov/data/
 ↪genomes/G
CF_000010065.1_ASM1006v1_genomic.fna}

{'assembly': '../../bioprov/data/genomes/GCF_000010065.1_ASM1006v1_genomic.
 ↪fna',
'taxon': 'Synechococcus elongatus PCC 6301'}
```

The `read_csv()` function also accepts these arguments and passes them to `from_df()`. So we can just do:

[6]:
```
proj = bp.read_csv(picocyano_dataset, file_cols=["assembly",],␣
 ↪tag="picocyano")

# Because of the path of the tutorials, we are going to have to process␣
 ↪the file,
# so we use from_df()

df = pd.read_csv(picocyano_dataset)
df['assembly'] = "../../" + df["assembly"]
proj = bp.from_df(df, file_cols="assembly", tag="W3C-PROV-tutorial.json");
 ↪ proj
```

[6]:
```
Project 'W3C-PROV-tutorial.json' with 5 samples
```

SeqFiles

The goal of BioProv is to make the provenance of biological data structures more accessible. To do this, there is the class **SeqFile**. An instance of **SeqFile** is a customization of **File** which also holds information about sequences, so it can load files such as FASTA and extract information using the BioPython modules `Bio.SeqIO` and `Bio.AlignIO`. When importing data, we can specify which columns have sequence files by using the `sequencefile_cols` argument. The `"assembly"` column of our dataset contains FASTA files, so we can load it as such, and use the `import_data=True` argument to extract information from the files.

[7]:
```
df = pd.read_csv(picocyano_dataset)
df['assembly'] = "../../" + df["assembly"]
proj = bp.from_df(df, sequencefile_cols="assembly",␣
 ↪tag="W3C-PROV-tutorial.json", import_data=True); proj
```

[7]:
```
Project 'W3C-PROV-tutorial.json' with 5 samples
```

SeqFiles possess attributes which are specific to biological sequences, such as number of sequences, number of basepairs, GC content, and N50. They are attributes of the **Sample** instance and also implemented in the **SeqStats** data class, which is also an attribute of the **SeqFile**.

[8]:
```
proj["GCF_000010065.1"].files['assembly'].__dict__
```

[8]:
```
{'path': PosixPath('/Users/vini/Bio/BioProv/docs/tutorials/../../bioprov/
 ↪data/ge
nomes/GCF_000010065.1_ASM1006v1_genomic.fna'),
 'name': 'GCF_000010065.1_ASM1006v1_genomic',
 'basename': 'GCF_000010065.1_ASM1006v1_genomic.fna',
 'directory':
PosixPath('/Users/vini/Bio/BioProv/docs/tutorials/../../bioprov/data/
 ↪genomes'),
 'extension': '.fna',
 'tag': 'assembly',
 'attributes': {},
 '_exists': True,
 '_size': '2.6 MB',
 'raw_size': 2730026,
 '_sha256':␣
 ↪'3a92f60662c48655d413ec017aaa2424128eb95882aa73005ec3bed266beba31',
 '_entity': None,
 'format': 'fasta',
```

```
 'records': {'NC_006576.1':␣
 ↪SeqRecord(seq=Seq('ATTTAAATCACTGGCATCAGCATTCGCAATATC
ATTGAGGTCAACAATACTTTC...GGC'), id='NC_006576.1', name='NC_006576.1',
description='NC_006576.1 Synechococcus elongatus PCC 6301 DNA, complete␣
 ↪genome',
dbxrefs=[])},
 '_generator': <Bio.SeqIO.FastaIO.FastaIterator at 0x7fdfbf427a00>,
 '_seqstats': SeqStats(number_seqs=1, total_bps=2696255, mean_bp=2696255.
 ↪0,
min_bp=2696255, max_bp=2696255, N50=2696255, GC=0.55484),
 '_parser': 'seq',
 '_max_seq': None,
 '_min_seq': None,
 'number_seqs': 1,
 'total_bps': 2696255,
 'mean_bp': 2696255.0,
 'min_bp': 2696255,
 'max_bp': 2696255,
 'N50': 2696255,
 'GC': 0.55484}
```

[9]:
```
proj["GCF_000010065.1"].files['assembly'].seqstats
```

[9]:
```
SeqStats(number_seqs=1, total_bps=2696255, mean_bp=2696255.0, min_-
bp=2696255,
max_bp=2696255, N50=2696255, GC=0.55484)
```

PresetPrograms

Now that we've seen an easier way to import data into BioProv, let us see an easier way to run **Programs**. BioProv has the class **PresetProgram**, which is an easier way to create Programs which will be run a lot. There are functions to call PresetPrograms in the `bioprov.programs` module. For this example, we are going to run the program Prodigal using a PresetProgram. Prodigal is a gene-calling software which predicts coding sequences from prokaryotic genomes.

[10]:
```
from bioprov.programs import prodigal

for sample in proj:
    sample.add_programs(prodigal(sample))
    sample.run_programs()
```

```
Updating file proteins with value /Users/vini/Bio/BioProv/docs/tutorials/..
 ↪/../b
ioprov/data/genomes/GCF_000010065.1_ASM1006v1_genomic_proteins.faa.
Updating file genes with value /Users/vini/Bio/BioProv/docs/tutorials/../..
 ↪/biop
rov/data/genomes/GCF_000010065.1_ASM1006v1_genomic_genes.fna.
Updating file scores with value /Users/vini/Bio/BioProv/docs/tutorials/../.
 ↪./bio
prov/data/genomes/GCF_000010065.1_ASM1006v1_genomic_scores.cds.
Running program 'prodigal' for sample GCF_000010065.1.
Command is:
/Users/vini/anaconda3/envs/bp-use-case/bin/prodigal \
        -i /Users/vini/Bio/BioProv/docs/tutorials/../../bioprov/data/
 ↪genomes/GCF
_000010065.1_ASM1006v1_genomic.fna \
        -a /Users/vini/Bio/BioProv/docs/tutorials/../../bioprov/data/
 ↪genomes/GCF
_000010065.1_ASM1006v1_genomic_proteins.faa \
        -d /Users/vini/Bio/BioProv/docs/tutorials/../../bioprov/data/
 ↪genomes/GCF
_000010065.1_ASM1006v1_genomic_genes.fna \
        -s /Users/vini/Bio/BioProv/docs/tutorials/../../bioprov/data/
 ↪genomes/GCF
_000010065.1_ASM1006v1_genomic_scores.cds
Updating file proteins with value /Users/vini/Bio/BioProv/docs/tutorials/..
 ↪/../b
ioprov/data/genomes/GCF_000007925.1_ASM792v1_genomic_proteins.faa.
Updating file genes with value /Users/vini/Bio/BioProv/docs/tutorials/../..
 ↪/biop
rov/data/genomes/GCF_000007925.1_ASM792v1_genomic_genes.fna.
Updating file scores with value /Users/vini/Bio/BioProv/docs/tutorials/../.
 ↪./bio
prov/data/genomes/GCF_000007925.1_ASM792v1_genomic_scores.cds.
Running program 'prodigal' for sample GCF_000007925.1.
Command is:
/Users/vini/anaconda3/envs/bp-use-case/bin/prodigal \
        -i /Users/vini/Bio/BioProv/docs/tutorials/../../bioprov/data/
 ↪genomes/GCF
_000007925.1_ASM792v1_genomic.fna \
        -a /Users/vini/Bio/BioProv/docs/tutorials/../../bioprov/data/
 ↪genomes/GCF
```

```
_000007925.1_ASM792v1_genomic_proteins.faa \
        -d /Users/vini/Bio/BioProv/docs/tutorials/../../bioprov/data/
 ↪genomes/GCF
_000007925.1_ASM792v1_genomic_genes.fna \
        -s /Users/vini/Bio/BioProv/docs/tutorials/../../bioprov/data/
 ↪genomes/GCF
_000007925.1_ASM792v1_genomic_scores.cds
Updating file proteins with value /Users/vini/Bio/BioProv/docs/tutorials/..
 ↪/../b
ioprov/data/genomes/GCF_000316515.1_ASM31651v1_genomic_proteins.faa.
Updating file genes with value /Users/vini/Bio/BioProv/docs/tutorials/../..
 ↪/biop
rov/data/genomes/GCF_000316515.1_ASM31651v1_genomic_genes.fna.
Updating file scores with value /Users/vini/Bio/BioProv/docs/tutorials/../.
 ↪./bio
prov/data/genomes/GCF_000316515.1_ASM31651v1_genomic_scores.cds.
Running program 'prodigal' for sample GCA_000316515.1.
Command is:
/Users/vini/anaconda3/envs/bp-use-case/bin/prodigal \
        -i /Users/vini/Bio/BioProv/docs/tutorials/../../bioprov/data/
 ↪genomes/GCF
_000316515.1_ASM31651v1_genomic.fna \
        -a /Users/vini/Bio/BioProv/docs/tutorials/../../bioprov/data/
 ↪genomes/GCF
_000316515.1_ASM31651v1_genomic_proteins.faa \
        -d /Users/vini/Bio/BioProv/docs/tutorials/../../bioprov/data/
 ↪genomes/GCF
_000316515.1_ASM31651v1_genomic_genes.fna \
        -s /Users/vini/Bio/BioProv/docs/tutorials/../../bioprov/data/
 ↪genomes/GCF
_000316515.1_ASM31651v1_genomic_scores.cds
Updating file proteins with value /Users/vini/Bio/BioProv/docs/tutorials/..
 ↪/../b
ioprov/data/genomes/GCF_000012625.1_ASM1262v1_genomic_proteins.faa.
Updating file genes with value /Users/vini/Bio/BioProv/docs/tutorials/../..
 ↪/biop
rov/data/genomes/GCF_000012625.1_ASM1262v1_genomic_genes.fna.
Updating file scores with value /Users/vini/Bio/BioProv/docs/tutorials/../.
 ↪./bio
prov/data/genomes/GCF_000012625.1_ASM1262v1_genomic_scores.cds.
Running program 'prodigal' for sample GCF_000012625.1.
```

```
Command is:
/Users/vini/anaconda3/envs/bp-use-case/bin/prodigal \
        -i /Users/vini/Bio/BioProv/docs/tutorials/../../bioprov/data/
 ↪genomes/GCF
_000012625.1_ASM1262v1_genomic.fna \
        -a /Users/vini/Bio/BioProv/docs/tutorials/../../bioprov/data/
 ↪genomes/GCF
_000012625.1_ASM1262v1_genomic_proteins.faa \
        -d /Users/vini/Bio/BioProv/docs/tutorials/../../bioprov/data/
 ↪genomes/GCF
_000012625.1_ASM1262v1_genomic_genes.fna \
        -s /Users/vini/Bio/BioProv/docs/tutorials/../../bioprov/data/
 ↪genomes/GCF
_000012625.1_ASM1262v1_genomic_scores.cds
Updating file proteins with value /Users/vini/Bio/BioProv/docs/tutorials/..
 ↪/../b
ioprov/data/genomes/GCF_000011485.1_ASM1148v1_genomic_proteins.faa.
Updating file genes with value /Users/vini/Bio/BioProv/docs/tutorials/../..
 ↪/biop
rov/data/genomes/GCF_000011485.1_ASM1148v1_genomic_genes.fna.
Updating file scores with value /Users/vini/Bio/BioProv/docs/tutorials/../.
 ↪./bio
prov/data/genomes/GCF_000011485.1_ASM1148v1_genomic_scores.cds.
Running program 'prodigal' for sample GCF_000011485.1.
Command is:
/Users/vini/anaconda3/envs/bp-use-case/bin/prodigal \
        -i /Users/vini/Bio/BioProv/docs/tutorials/../../bioprov/data/
 ↪genomes/GCF
_000011485.1_ASM1148v1_genomic.fna \
        -a /Users/vini/Bio/BioProv/docs/tutorials/../../bioprov/data/
 ↪genomes/GCF
_000011485.1_ASM1148v1_genomic_proteins.faa \
        -d /Users/vini/Bio/BioProv/docs/tutorials/../../bioprov/data/
 ↪genomes/GCF
_000011485.1_ASM1148v1_genomic_genes.fna \
        -s /Users/vini/Bio/BioProv/docs/tutorials/../../bioprov/data/
 ↪genomes/GCF
_000011485.1_ASM1148v1_genomic_scores.cds
```

Because Prodigal is a **PresetProgram**, it already expects a Sample to have a File tagged as `"assembly"`. This can be customized by setting the `input_files` argument

in the constructor of PresetProgram:

[11]:
```
bp.PresetProgram?
```

```
Init signature:
bp.PresetProgram(
    name=None,
    params=None,
    sample=None,
    input_files=None,
    output_files=None,
    preffix_tag=None,
    extra_flags=None,
)
Docstring:
Class for holding a preset program and related functions.

A WorkflowStep instance inherits from Program and consists of an instance
of Program with an associated instance of Sample or Project.
Init docstring:
:param name: Instance of bioprov.Program
:param params: Dictionary of parameters.
:param sample: An instance of Sample or Project.
:param input_files: A dictionary consisting of Parameter keys as keys and␣
 ↪a File.tag
                    as value, where File.tag is a string that must be a␣
 ↪key in
                    self.sample.files with a corresponding existing file.
:param output_files: A dictionary consisting of Parameter keys as keys and␣
 ↪a tuple
                     consisting of (File.tag, suffix) as value.
                     File.tag will become a key in self.sample.files and␣
 ↪the its value
                     will be the sample_name + suffix.
:param preffix_tag: A value in the input_files argument, which corresponds
                    to a key in self.sample.files. All file names of output
                    files will be stemmed from this file, hence 'preffix'.
:param extra_flags: A list of command line parameters (strings), known as␣
 ↪flags or
                    switches, to add to the program's command.
File:           ~/anaconda3/envs/bp-use-case/lib/python3.8/site-packages/
 ↪bioprov/src/main.py
```

```
Type:           type
Subclasses:
```

BioProvProjects and W3C-PROV documents (under development)

Now that we've imported our **Project** and ran the **PresetProgram** Prodigal on the **Samples**, we can record the provenance from our workflow. For this, we use the **BioProvDocument** class, which couples the **Project** to a **ProvDocument** from the Prov library.

[12]:
```
prov = bp.BioProvDocument(proj)
```

Now, our **Project** has been associated with a `prov.ProvDocument` object, which is an attribute of the **BioProvDocument** instance. Although we have overwritten the `proj` variable, our project is still accessible as the `.project` attribute of the BioProvDocument:

[13]:
```
print(prov.project)
```

```
Project 'W3C-PROV-tutorial.json' with 5 samples
```

There are numerous ways to manipulate this ProvDocument.

- We can extract the PROV-N format, which is a human-readable provenance format, by using the `get_provn()` method;
- We can serialize the document as a W3C-PROV compatible JSON
- We can export the document as a provenance graph using `prov_to_dot()`

[14]:
```
print("PROV-N", "\n\n", prov.ProvDocument.get_provn()[:1000])
print("PROV-JSON", "\n\n", prov.ProvDocument.serialize()[:1000])
dot = prov.dot
dot.write_png("W3C-PROV-tutorial.png")
# You can also save the provenance graph in DOT language
dot.write_raw("W3C-PROV-tutorial.gv")
```

```
PROV-N

 document
  prefix project <Project 'W3C-PROV-tutorial.json' with 5 samples>
  prefix users <Users associated with BioProv Project 'W3C-PROV-tutorial.
↪json'>
  prefix samples <Samples associated with bioprov Project 'W3C-PROV-
tutorial.json'>
```

```
  bundle users:vini
    default <vini>
    prefix envs <Environments associated with User 'vini'>
    prefix users <Users associated with BioProv Project 'W3C-PROV-
tutorial.json'>

    agent(users:vini)
    agent(envs:
 ↪c4c5922f6c7d7c0206378e3708ab74fe4c44d841dec96549484bc3842333bed8)
    actedOnBehalfOf(envs:
 ↪c4c5922f6c7d7c0206378e3708ab74fe4c44d841dec96549484bc38
42333bed8, users:vini, -)
  endBundle
  bundle project:Project 'W3C-PROV-tutorial.json' with 5 samples
    default <W3C-PROV-tutorial.json>
    prefix files <Files associated with Project W3C-PROV-tutorial.json>
    prefix programs <Programs associated with Project W3C-PROV-tutorial.
 ↪json>

    entity(project:Project 'W3C-PROV-tutorial.json' with 5 samples)
  endBundle
  bundle sampl
```

The generated figure illustrates provenance relationships between objects.[2]

[2] https://github.com/vinisalazar/BioProv/blob/dev/docs/tutorials/W3C-PROV-tutorial.png

Apêndice C

Knowledge Management in Genomics: The Role of Data Provenance

O resumo a seguir foi desenvolvido como parte dessa dissertação e foi aceito para apresentação de pôster no evento X-Meeting 2019, na categoria de "Databases and Software". O pôster está disponível no seguinte endereço: https://drive.google.com/file/d/1B_xzXjYh9NvkEaqR_tKPT3AEe0AObAXC/

Genomics as a discipline has grown considerably in recent years and its methods have proven to be helpful for solving diverse problems across various domains of the life sciences. It has been constantly driven by data life cycles, with scientists relying on public, curated data repositories to perform their own experiments. The consequences of this are that the issue of managing data correctly and efficiently have been central to the field, and with the exponential growth in publicly available data, this has turned into a "big data issue" which concerns individual scientists, research groups, institutions, and international consortia. We discuss how knowledge management (KM) can address some of the big data issues in genomics. The KM approach has proven to be valuable in case studies, due to its systematic process of defining workflows that improves efficiency and reproducibility of experiments. However, combining a KM workflow with genomics has challenges related to the genomics data life cycle like provenance. Tracking the provenance of data in genomics is becoming part of its data life cycle, but provenance data is not part of the KM workflow. Effective data management, good practices in scientific computing, FAIR data sharing, and collaborative work between experts in different disciplines can all be facilitated by provenance tracking. Provenance tracking is a key aspect in establishing the data-information-knowledge cycle, which can be used as a framework for planning, executing scientific and monitoring experiments. Because prove-

nance is information providing context to data, it plays a fundamental role in generating and sharing knowledge at the end of the cycle. Provenance tracking needs data capture and storage, which may affect the performance of the genomics workflow. Our proposal is focused on bioinformatics workflows and how they should be planned from a KM provenance-based perspective. We show how to adopt a KM provenance-based perspective of experiments in genomics with negligible computational costs by adopting a dataflow analysis system, in this case, the DfAnalyzer tool, which adopts the W3C PROV standard. By coupling DfAnalyzer with a genomic bioinformatics pipeline, it is possible to capture provenance data which when queried generates information about trade-offs, for example, of performance and sensitivity, providing the research with an enriched knowledge of his analysis. The different DfAnalyzer components will align with those of knowledge management: the Provenance Data Extractor, Raw Data Extractor and Raw Data Index will act on data capture and storage, the Query Interface and Dataflow Viewer on information summarizing and analysing, allowing a synthesis of knowledge around the experiment which can support decision making. This is particularly useful when scaling experiments with a large number of samples or parameter sweeping. For demonstration, we executed DfAnalyzer with a bacterial genome annotation and phylogenetic analysis workflow, using the DfA Python library. We considered the steps of data and metadata collection, sequence statistics, gene call predictions, annotation with the NCBI COG database, and pairwise Average Nucleotide Identity between genomes, integrating each step to the DfA Python library with a custom Python package. In our workflow case study we show the benefits of monitoring, querying and reuse of methods and results in genomics experiments, showcasing how a KM approach to research in the field may as well be a part of the new paradigm of data-intensive discovery in biology.