Workflow para Alinhamento Exato de Sequências em Sistemas de Processamento de Alto Desempenho

Rafael Terra[1], Kelen Souza[1,2], Hiago Rocha[1], Carla Osthoff[1], Diego Carvalho[3], Kary Ocaña [1]

[1]Laboratório Nacional de Computação Científica (LNCC)

{rafaelst, kelenbs, mayk, osthoff, karyann}@lncc.br

[2]Faculdade de Educação Tecnológica do Estado do Rio de Janeiro (FAETERJ)

[3]Centro Federal de Educação Tecnológica Celso Suckow da Fonseca (CEFET-RJ)

diego.carvalho@cefet-rj

Resumo. *O Alinhamento Múltiplo de Sequências (AMS) é uma etapa fundamental na biologia evolutiva molecular, com impacto direto na identificação de marcadores associados a doenças genéticas e infecciosas. A qualidade dos alinhamentos é determinante para a confiabilidade das interpretações biológicas. No entanto, algoritmos exatos para AMS enfrentam limitações quanto ao número de sequências que podem ser processadas, mesmo em ambientes de Processamento de Alto Desempenho (PAD), devido à natureza NP-difícil do problema. Assim, a prática usual envolve a seleção manual de subconjuntos reduzidos de sequências. Neste trabalho, propomos e avaliamos um workflow científico em PAD para AMS com algoritmos exatos, incorporando a seleção automática do subconjunto representativo de sequências, de modo a contornar as restrições das ferramentas disponíveis. Os experimentos, considerando implementações do workflow com PyCOMPSs e com scripts Shell, apresentaram ganhos de* $2,08\times$ *e* $3,95\times$*, respectivamente, em relação à execução sequencial das tarefas. Em particular, a implementação com PyCOMPSs mostrou melhor escalabilidade, alcançando 80,8% de ganho no alinhamento de 38 sequências.*

1. Introdução

O Alinhamento Múltiplo de Sequências (AMS), definido pelo alinhamento de três ou mais sequências gênicas, constitui uma etapa essencial na biologia evolutiva molecular, servindo de base para análises como reconstruções filogenéticas, detecção de seleção positiva e estudos em epidemiologia molecular [Yang 2007, Schabauer et al. 2012, Gao et al. 2019]. No entanto, o AMS é um problema NP-difícil, de modo que estratégias voltadas à obtenção do alinhamento ótimo demandam elevado poder computacional e grande capacidade de memória, o que restringe sua aplicação a um número reduzido de sequências (10 a 20 elementos de comprimento moderado, devido ao crescimento exponencial do espaço de busca) [Slowinski 1998], mesmo quando executado em ambientes de Processamento de Alto Desempenho (PAD) [Sundfeld et al. 2018].

Contudo, com o crescimento exponencial dos dados genômicos, impulsionado pelo avanço das tecnologias de sequenciamento de nova geração (NGS, do inglês *Next-Generation Sequencing*), pesquisadores frequentemente lidam com grandes conjuntos de

sequências biológicas, que podem variar de milhares a milhões de elementos em bancos de dados públicos [Tripathi et al. 2016]. Em cenários nos quais se deseja empregar algoritmos exatos, torna-se necessário adaptar o conjunto de dados inicial às limitações no número de sequências impostas por tais métodos, geralmente por meio da seleção de um subconjunto representativo das sequências originais, permitindo reduzir o problema a uma escala computacionalmente tratável.

Entretanto, a escolha dessas sequências não é trivial, pois envolve decisões sobre critérios de representatividade, diversidade e relevância biológica, bem como sobre o próprio processo de seleção, que, quando realizado manualmente, pode ser trabalhoso, suscetível a vieses e propenso a erros, dada a complexidade dos experimentos. Nesse contexto, torna-se essencial investigar abordagens que automatizem esse processo, garantindo que o AMS possa ser executado em ambientes de PAD, ao mesmo tempo em que se aumenta a acurácia e se reduz o tempo necessário para a obtenção dos alinhamentos.

Para tornar esse processo mais acessível a pesquisadores que frequentemente não possuem familiaridade com ferramentas e ambientes de PAD, os Sistemas de Gerenciamento de *Workflows* Científicos (SWfMS, do inglês *Scientific Workflow Management Systems*) surgem como abordagens eficazes. Essas ferramentas permitem modelar, executar, monitorar e reproduzir experimentos computacionais complexos de forma estruturada e automatizada [Cohen-Boulakia et al. 2017]. Além de viabilizarem a execução paralela e escalável de *pipelines* computacionais, tornando-os particularmente adequados para análises evolutivas em larga escala [Yang 2007], os SWfMS também automatizam o gerenciamento de tarefas e dados intermediários associados à execução do *workflow*, reduzindo a necessidade de intervenção manual.

Com base nos aspectos previamente discutidos, neste trabalho propomos a modelagem de um *workflow* científico para o AMS, capaz de processar um grande número de sequências em sistemas de PAD. Para acelerar a seleção de sequências e a obtenção do alinhamento ótimo, nosso *workflow* aplica uma estratégia em duas fases: **Fase AS,** considera-se um subproblema do AMS em que se utiliza um algoritmo polinomial exato, o alinhamento par-a-par, para executar múltiplos alinhamentos simultaneamente nos diferentes recursos de processamento disponíveis no ambiente de PAD, gerando os *scores* correspondentes a cada par de sequências; **Fase AMS,** com base nos *scores* gerados na fase anterior, o *workflow* seleciona um subconjunto de sequências relevantes – que podem ser similares ou dissimilares, conforme a estratégia escolhida pelo usuário – e realiza o AMS utilizando um algoritmo exato.

Em resumo, as principais contribuições deste trabalho são:

- A **estratégia** que combina o alinhamento par-a-par para seleção de subconjuntos de sequências relevantes e o AMS por meio de algoritmos exatos, respeitando as limitações de entrada dos algoritmos de AMS exatos;
- A **modelagem** da estratégia no SWfMS PyCOMPSs, possibilitando a execução e gerenciamento de forma transparente, escalável e reprodutível em ambientes de PAD.

Com base em nossos resultados experimentais, considerando execuções com 9 a 38 sequências (com tamanho médio de 1.483 nucleotídeos) em um processador de 48 núcleos, observou-se que as versões do *workflow* desenvolvidas com PyCOMPSs e com

scripts Shell obtiveram acelerações de $2,08\times$ e $3,95\times$, respectivamente, quando comparadas à execução sequencial convencional, na qual cada tarefa é executada de forma sucessiva. Em especial, a abordagem baseada em PyCOMPSs apresentou maior escalabilidade, atingindo um ganho de 80,8% no processamento de 38 sequências.

O presente trabalho está organizado da seguinte forma: A Seção 2 discute os fundamentos teóricos e o estado da arte relacionado ao AMS e ao uso de SWfMS em ambientes de PAD. A Seção 3 descreve em detalhes o *workflow* proposto, incluindo sua arquitetura, estratégias de combinação e aspectos de implementação. A Seção 4 apresenta a metodologia experimental, enquanto a Seção 5 mostra análise dos resultados obtidos. Por fim, a Seção 6 discute as conclusões e propõe direções para trabalhos futuros.

2. Fundamentação Teórica e Trabalhos Relacionados

Esta seção apresenta os conceitos fundamentais necessários para a compreensão do artigo, bem como os trabalhos relacionados.

2.1. Alinhamento Múltiplo de Sequência

O **Alinhamento de Sequência (AS)** consiste em organizar duas sequências biológicas (DNA, RNA ou proteínas), inserindo lacunas ('–') quando necessário, de modo a maximizar regiões de similaridade. Isso permite identificar homologias, mutações pontuais e eventos de inserção/deleção. Em sua forma mais simples, AS par-a-par, tanto o alinhamento *global*, (i.e., sequências inteiras do início ao fim) [Needleman and Wunsch 1970] quanto o alinhamento *local* (i.e., encontra o melhor trecho local de similaridade dentro das sequências) [Smith et al. 1981] podem ser obtidos em tempo polinomial via programação dinâmica: $O(mn)$, onde m e n são os comprimentos das sequências. Entretanto, variantes mais amplas do AS para alinhar três ou mais sequências biológicas – usadas em aplicações como inferência de árvores filogenéticas, identificação de domínios conservados e predição de estruturas 3D – exigem o **Alinhamento Múltiplo de Sequências (AMS)**. Esse problema é **NP-difícil**, pois o número de alinhamentos cresce exponencialmente com o número de sequências e seu comprimento [Wang and Jiang 1994].

Formulação do Problema de AMS. Sejam $S_1, S_2, \ldots, S_n$ sequências sobre um alfabeto Σ, com comprimentos $L_1, L_2, \ldots, L_n$. Queremos encontrar sequências estendidas $\tilde{S}_i$ (com lacunas '–' inseridas) todas de tamanho L, satisfazendo:

(*i*) Cada $\tilde{S}_i$ é obtida de S_i apenas por inserções de lacunas.
(*ii*) O comprimento comum é L, de modo que $|\tilde{S}_i| = L \; \forall i$.

Define-se então a pontuação *Sum-of-Pairs (SP-score)* como:

$$\mathrm{SP}(\tilde{S}_1, \ldots, \tilde{S}_n) = \sum_{1 \le i < j \le n} \sum_{k=1}^{L} \delta(\tilde{S}_i[k],\, \tilde{S}_j[k]),$$

em que δ é uma função de pontuação que atribui $+1$ para *match* e -1 como penalidade para *mismatches* e lacunas. Note que um *SP-score* mais alto indica um alinhamento mais conservado entre as sequências. Por tanto, o objetivo é encontrar o alinhamento $\{\tilde{S}_i\}$ de máximo *SP-score*:

$$\max_{\tilde{S}_1, \ldots, \tilde{S}_n} \mathrm{SP}(\tilde{S}_1, \ldots, \tilde{S}_n).$$

Complexidade Computacional. Como mencionado anteriormente, o problema de AMS é NP-difícil, uma vez que seu espaço de busca cresce exponencialmente com o número de sequências n e o comprimento L das mesmas, como demonstrado por Wang e Jiang (1994). Esse crescimento exponencial torna proibitivo achar o alinhamento ótimo para entradas muito grandes. Além disso, estudos combinatórios ilustram que, para cinco sequências de DNA, cada uma com cinco nucleotídeos, existem aproximadamente $1,05 \times 10^{18}$ alinhamentos possíveis [Slowinski 1998]. Vale ressaltar que esse número foi obtido considerando um conjunto relativamente pequeno de sequências, muito menores e mais curtas do que aquelas frequentemente utilizadas por filogeneticistas moleculares, evidenciando o crescimento exponencial do tamanho do problema [Tripathi et al. 2016].

2.2. Estratégias de Otimização

Diversas abordagens para o AMS têm sido propostas na literatura, baseando-se em estratégias **heurísticas** [Sievers and Higgins 2018, Lassmann 2020, Katoh and Standley 2013] ou **exatas** [Papadopoulos and Agarwala 2007, De O. Sandes et al. 2016, Sundfeld et al. 2018]. Entretanto, essas abordagens diferem quanto ao objetivo na busca por soluções biologicamente relevantes: heurísticas privilegiam a eficiência computacional e a rapidez na geração dos resultados, ao passo que métodos exatos priorizam a maximização da acurácia dos alinhamentos, sendo mais adequados em cenários que demandam alta confiabilidade nas interpretações biológicas.

Apesar da existência de diferentes abordagens exatas na literatura que tratam o problema de diversas formas – por exemplo, por meio de *branch-and-cut* [Prestwich et al. 2003], programação baseada em restrições [Papadopoulos and Agarwala 2007], *Block Pruning* [De O. Sandes et al. 2016] e o algoritmo A^* [Sundfeld et al. 2018], a seguir nos concentramos na descrição das abordagens propostas por De O. Sandes et al. (2016) e por Sundfeld et al. (2018), que operam de forma integrada no contexto do nosso *workflow* (ver seção 3).

MASA. A MASA [De O. Sandes et al. 2016] é uma ferramenta que implementa os algoritmos de Needleman–Wunsch (alinhamento global) e Smith–Waterman (alinhamento local), para alinhamento de sequências par-a-par, utilizando programação dinâmica (PD). Para reduzir consumo de memória, a ferramenta adota a estratégia de Myers-Miller, o que permite reconstruir o alinhamento com complexidade de espaço linear em relação ao tamanho das sequências. Para possibilitar paralelismo escalável na construção da matriz de PD, o MASA emprega a técnica de *Block Pruning*, que funciona da seguinte forma: (*i*) a matriz de PD é dividida em blocos regulares de tamanho fixo, e.g., 1024×1024 células; (*ii*) cada bloco é avaliado com base em um limite inferior de pontuação possível; (*iii*) se esse limite indicar que nenhum alinhamento ótimo pode passar pelo bloco, ele é completamente descartado, evitando cálculos desnecessários. Isso reduz significativamente o número de células processadas, especialmente em sequências com alta similaridade, acelerando o alinhamento sem comprometer a qualidade biológica. Além disso, o MASA paraleliza o cálculo por meio do OpenMP, mapeando o processamento de blocos para múltiplas *threads*. Cada *thread* é responsável por computar um ou mais blocos viáveis, de forma a explorar ao máximo os núcleos disponíveis.

PA-Star. A ferramenta PA-Star [Sundfeld et al. 2018] implementa uma estratégia paralela baseada no algoritmo exato A^* para o problema de AMS. O A^* é uma técnica

de busca informada que combina a exploração sistemática do espaço de busca com uma função heurística *admissível*, ou seja, que nunca superestima o custo real restante até o objetivo – garantindo, assim, a obtenção da solução ótima. A função de avaliação utilizada é dada por $f(n) = g(n) + h(n)$, onde $g(n)$ representa o custo acumulado desde o estado inicial até o nó n, e $h(n)$ é a heurística que estima o custo do caminho restante. No contexto do AMS, o PA-Star modela o espaço de alinhamento como um grafo multidimensional acíclico, em que cada nó corresponde a uma configuração parcial de alinhamento das sequências. Para lidar com a complexidade computacional, a ferramenta explora *paralelismo intrínseco* na expansão dos nós e no cálculo das funções de custo, distribuindo as tarefas sobre múltiplos núcleos de processamento – o paralelismo é implementado através de Pthreads. Apesar da paralelização, o PA-Star continua limitado por sua complexidade exponencial em relação ao número de sequências k, o que restringe sua aplicabilidade prática a conjuntos pequenos (tipicamente $k \leq 7$, dependendo do comprimento das sequências). Ainda assim, sua utilização é altamente relevante em contextos onde a qualidade biológica do alinhamento é crítica.

2.3. *Workflows* Científicos

Em ambientes de PAD, experimentos científicos podem ser representados por abstrações conhecidas como *workflows* científicos. Esses *workflows* são modelos computacionais que descrevem, organizam e automatizam a execução de experimentos compostos por múltiplas tarefas interdependentes. A execução desses *workflows* é realizada por meio *Scientific Workflow Management Systems* (SWfMS), que orquestram todas as atividades segundo as dependências definidas previamente. Há diversos SWfMS e *frameworks* para essa finalidade [Suter et al. 2025]. Contudo, neste trabalho, o foco será o *framework* PyCOMPSs [Tejedor et al. 2017].

PyCOMPSs. O PyCOMPSs é um *framework* construído sobre o sistema COMPSs, projetado para o desenvolvimento de *workflows* científicos em Python. Ele facilita o processamento de grandes volumes de dados e supera as limitações do *multithreading* no Python, oferecendo paralelismo distribuído e assíncrono [Tejedor et al. 2017]. Geralmente, um *workflow* em PyCOMPSs é escrito como um *script* Python convencional. No entanto, funções que devem ser executadas de forma assíncrona são marcadas com o decorador `task`. Esse decorador converte a função em uma tarefa distribuída, permitindo que o *runtime* construa um grafo de dependências (DAG), explore o paralelismo implícito e gerencie automaticamente as execuções e transferências de dados entre nós.

3. Modelagem do *Workflow*

O presente *workflow* foi concebido com o objetivo de realizar a seleção e o alinhamento múltiplo exato de um subconjunto de sequências genéticas. Para tanto, ele aplica uma estratégia em duas fases que: (***Fase AS***) considera um subproblema do AMS em que se utiliza um algoritmo polinomial exato para o alinhamento par-a-par – algoritmo Needleman–Wunsch implementado pela ferramenta MASA; e (***Fase AMS***) seleciona um subconjunto de sequências relevantes – que podem ser similares ou dissimilares, conforme a estratégia definida pelo usuário ao início da execução do *workflow* – e realiza o AMS utilizando um algoritmo exato A^* implementado pela ferramenta PA-Star.

Tanto MASA quanto PA-Star implementam algoritmos exatos, conforme descrito na seção 2. Além disso, o *workflow* foi projetado para explorar poder computacional

de ambientes de PAD, permitindo o processamento de conjuntos de dados complexos, caracterizados por um grande número de sequências e/ou sequências muito longas. A seguir, descrevemos o fluxo de execução do *workflow*, também ilustrado na Figura 1.

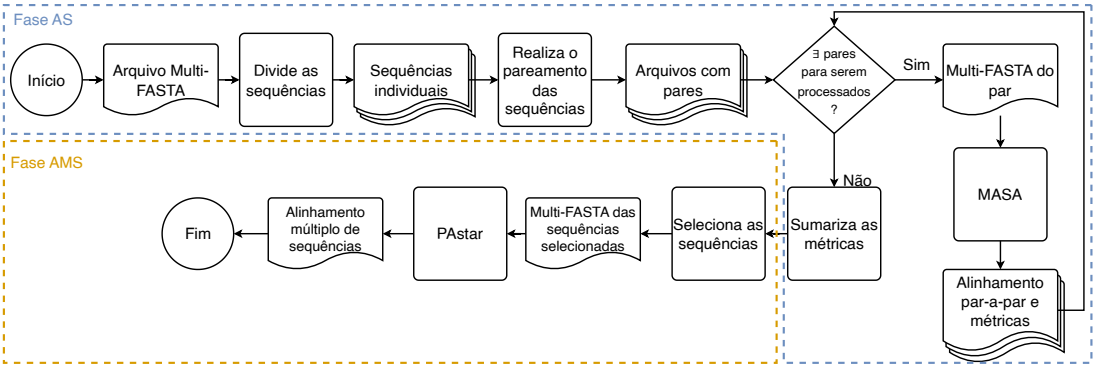


Figura 1. Diagrama detalhado do *workflow*.

3.1. Fase AS

Esta fase marca o ***Início*** da execução do *Workflow*. Utilizando como entrada o formato de ***Arquivo Multi-FASTA***[1] contendo diversas sequências genéticas, o *workflow* ***Divide as sequências*** por meio da biblioteca BioPython[2], escolhida por ser desenvolvida em *Python*, dispensar a necessidade de programas externos e permitir, de forma rápida, a leitura do arquivo de entrada e a geração de arquivos individuais para cada sequência por meio de funções nativas da linguagem. Em seguida, o *workflow* ***Realiza o pareamento das sequências*** através de uma combinação simples entre elas, gerando assim $[n \times (n-1)]/2$ pares (***Arquivos com pares***), onde n representa o número total de sequências no arquivo de entrada.

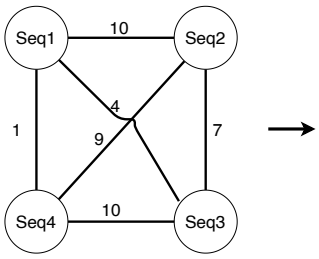
Para cada par, o *workflow* cria um diretório correspondente e armazena o arquivo em formato ***Multi-FASTA do par*** neste diretório. Após isso, o programa ***MASA*** é executado, utilizando como entrada o arquivo do par e resultando em um alinhamento par-a-par exato, acompanhado de um conjunto de métricas (*Alinhamento par-a-par, e métricas*) que avaliam a qualidade do alinhamento, entre elas: *SP-score* (ver seção 2), número de *matches* e número de *gaps*. Ao final desta etapa, o *workflow* ***Sumariza as métricas*** para todos os pares e ***Seleciona as sequências*** para realizar fase AMS.

3.2. Fase AMS

Essa fase se inicia com o procedimento de **Seleção das sequências** considerando o *SP-scores* obtidos a partir das métricas sumarizadas, assim como o parâmetro do usuário $p = \{S, D\}$, para escolha das sequências mais similares (S) ou mais divergentes (D). O procedimento de seleção funciona da seguinte forma: dado o grafo $G = (V, E)$, onde cada nó $v_i \in V$ representa uma sequência, e cada aresta $(v_i, v_j) \in E$ representa o alinhamento par-par entre as sequências v_i e v_j, conforme ilustrado na Figura 2. O peso associado a cada aresta corresponde à pontuação total de similaridade (*SP-score*) entre as respectivas

[1] Arquivo de texto onde cada sequência é representada por um conjunto de caracteres e é precedida por uma linha de cabeçalho com o identificador da sequência.

[2] `https://biopython.org/`



Classificação	Sequência	Valor
1º	Seq1	(10 + 4 +1)/3 = 5
2º	Seq4	(9 + 10 + 1)/3 = 6.66
3º	Seq3	(4 + 10 + 7)/3 = 7
4º	Seq2	(10 + 7 + 9)/3 = 8.66

Figura 2. Exemplo de ordenação de um conjunto de alinhamentos par-a-par.

sequências. Dessa forma, o grafo gerado é completo, ou seja, $\forall\ v_i, v_j \in V,\ i \neq j$, tem-se que $\exists\ v_i, v_j \in E$.

Seguindo essa estrutura, o peso médio de cada nó é calculado como o somatório dos pesos das arestas que o conectam, dividido pelo número total de arestas incidentes, conforme ilustrado na Figura 2. Com base nesses valores, a seleção do subconjunto de sequências é realizada da seguinte forma: as sequências são ordenadas de acordo com os pesos médios, sendo selecionadas, por padrão, as k sequências com maior peso quando o objetivo for escolher as mais similares (S), ou as k sequências com menor peso quando se busca as mais divergentes (D), de acordo com a escolha do usuário – critério definido como parâmetro no início da execução do *workflow*.

A partir do arquivo ***Multi-FASTA das sequências selecionadas***, o *workflow* executa o programa ***PA-Star*** utilizando o número de *threads* especificado no início da execução do *workflow*. Como resultado, o programa gera o ***Alinhamento múltiplo das sequências*** em um arquivo no formato FASTA, finalizando a execução do *workflow* (***Fim***).

3.3. Detalhes de Implementação

Nós implementamos o *workflow* utilizando a biblioteca PyCOMPSs, responsável pela orquestração paralela das tarefas. Como alternativa, também desenvolvemos uma versão baseada em *scripts* Shell e Python, com integração direta aos binários dos programas MASA e PA-Star. O código-fonte do *workflow* é aberto e está disponível no GitHub: `https://github.com/kelen-souza/Workflow-MASAOpenMP-PAStar`. O repositório disponibiliza instruções para a execução, além dos dados utilizados nos experimentos descritos nas Seções 4 e 5.

4. Metodologia

***Dataset*.** Nossos experimentos utilizaram como entrada um *dataset* composto por sequências do gene *E* do vírus da Dengue, obtidas do GenBank[3]. As sequências foram organizadas em quatro grupos de diferentes tamanhos, com o objetivo de avaliar a escalabilidade do *workflow* frente a conjuntos de dados de diferentes tamanhos: 9, 18, 28 e 38 sequências, com comprimento médio de ≈ 1484 pares de bases.

Ferramentas e Bibliotecas. O *workflow* foi implementado utilizando as seguintes versões de *software*: Python 3.11.7, PyCOMPSs 3.3.3, MASA OpenMP 1.0.1.1024, PA-Star2 v2.0 e BioPython 1.85. As execuções foram realizadas por meio do módulo *en-*

[3]`https://www.ncbi.nlm.nih.gov/genbank/`

queue_compss, integrante do ambiente PyCOMPSs, o qual permite a submissão e o gerenciamento de tarefas paralelas em ambientes de PAD.

Ambiente de Execução. Os experimentos foram executados no supercomputador Santos Dumont (SDumont) [4] utilizando um nó computacional equipado com 2× *Intel Xeon Cascade Lake Gold 6252* (24 núcleos físicos por *socket*), totalizando 48 núcleos físicos por nó (*hyperthreading* desabilitado), possui 384 GB de memória RAM e utiliza o sistema operacional Red Hat Enterprise Linux 8.8, utilizando o kernel Linux versão 4.18.

Estratégias Analisadas. Em nossos experimentos, avaliamos as duas versões do *workflow* (ver Seção 3), além de uma abordagem adicional usada como *baseline*, que representa a forma tradicional com que um biólogo poderia executar manualmente o MASA seguido do PA-Star para realizar o AMS:

- ***Workflow com PyCOMPSs:*** Execução completa do *workflow* com orquestração paralela utilizando a biblioteca PyCOMPSs.
- ***Workflow sem PyCOMPSs:*** Execução completa do *workflow* utilizando *scripts* Shell e Python, com chamadas diretas aos binários dos programas.
- ***Normal (baseline):*** Execução sequencial dos programas MASA e PA-Star, etapa por etapa, sem paralelismo nem encapsulamento das tarefas em um *workflow* automatizado.

Avaliação de Desempenho. As estratégias descritas anteriormente foram avaliadas a partir de arquivos de entrada contendo diferentes grupos de sequências (9, 18, 28 e 38). Para cada grupo, após a Fase AS, foram selecionadas cinco sequências para seguir para a Fase AMS. Essa quantidade de sequências foi escolhida por estar abaixo do limite imposto pela ferramenta PA-Star (vide Seção 5), mas apresentando maior complexidade em comparação ao caso mínimo (três sequências). Com isso, foram realizadas cinco execuções independentes e sumarizadas com a média aritmética. Os tempos de execução foram coletados por meio da ferramenta *sacct*, disponibilizada pelo sistema de gerenciamento de filas SLURM [Yoo et al. 2003].

5. Resultados

Esta seção apresenta os resultados e discussões acerca das análises realizadas no presente trabalho.

5.1. Comparação Geral

Nesta seção, analisamos os tempos de execução das duas versões do *workflow* (**com** e **sem** o uso do PyCOMPSs) em comparação com a abordagem ***Normal*** (ver Seção 4). A Figura 3 apresenta os tempos de execução (eixo *y*, em segundos) para cada estratégia avaliada, representadas por barras de cores distintas, à medida que o número de sequências de entrada aumenta (eixo *x*).

Com base na Figura 3, observa-se que, exceto para a entrada com nove sequências, ambas as versões do nosso *workflow* superam a abordagem ***Normal***, especialmente à medida que o número de sequências aumenta. Esse comportamento ocorre porque as versões do *workflow* exploram paralelismo e automação na execução das tarefas, reduzindo o

[4] `https://sdumont.lncc.br/`

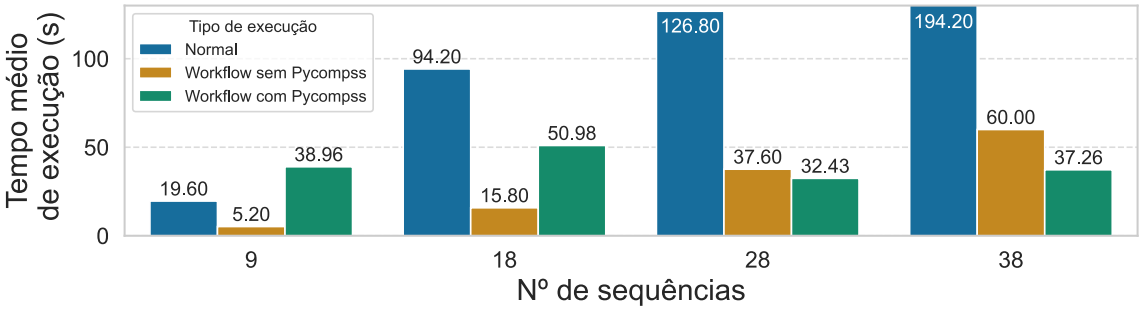


Figura 3. Tempo de execução das diferentes estratégias conforme o número de sequencias aumenta.

tempo ocioso entre etapas. Nessas condições, nosso *workflow* alcança um ganho de desempenho de $3,95\times$ (**sem** PyCOMPSs) e $2,08\times$ (**com** PyCOMPSs) na média geométrica dos diferentes tamanhos de entrada. Considerando o grupo de entradas com maior número de sequências (38 sequências), o *workflow* demonstra uma vantagem de $69,1\%$ e $80,8\%$ para as versões **sem** e **com** PyCOMPSs, respectivamente.

Ao compararmos apenas as duas versões do nosso *workflow*, observamos que: (*i*) para até 18 sequências, a versão **sem** PyCOMPSs apresenta menores tempos de execução; (*ii*) a partir de um maior número de sequências (28), a versão **com** PyCOMPSs passa a apresentar melhor desempenho. O comportamento descrito em (*i*) ocorre porque o tempo necessário para inicializar o ambiente PyCOMPSs representa uma parcela significativa do tempo total de execução em cenários menores (5 segundos em nossos experimentos). No entanto, à medida que o número de sequências aumenta, esse *overhead* torna-se proporcionalmente menos relevante, o que explica a inversão observada em (*ii*). Nesses casos, o uso do PyCOMPSs resulta em reduções no tempo de execução de aproximadamente $13,8\%$ para 28 sequências e $37,9\%$ para 38 sequências.

Adicionalmente, para ilustrar o alto nível de paralelismo permitido pelo *workflow*, a Figura 4 apresenta o DAG da execução do *workflow* com 9 sequências, gerado a partir dos dados do *logs* de execução do PyCOMPSs. Grafos semelhantes são observados para as execuções das demais sequências, com variações ocasionadas pelo número de pares. Nessa figura, cada círculo colorido representa uma tarefa que o *workflow* deve executar, e as linhas retas indicam as relações de dependência entre as tarefas. É importante destacar que todas as tarefas em uma mesma fronteira (por exemplo, azul, laranja, ou verde) podem ser executadas simultaneamente, por serem independentes entre si. Conforme demonstrado em nossos resultados, o *workflow* explora esse alto grau de paralelismo para iniciar as tarefas o mais cedo possível, utilizando de forma mais eficiente os recursos de PAD.

5.2. Analisando os Ganhos nas Fases *Workflow*

Esta seção apresenta uma análise individual das fases do *workflow*, com o objetivo de evidenciar os ganhos de desempenho alcançados em cada uma delas.

Fase AS. Na implementação desta fase, consideramos duas estratégias distintas: (*Paralelo*) executar múltiplas instâncias do MASA de forma paralela em um único nó computacional, uma instância após a outra, aproveitando o paralelismo no nível de alinhamentos par-a-par; ou (*Sequencial*) executar várias instâncias do MASA em modo *single-threaded*,

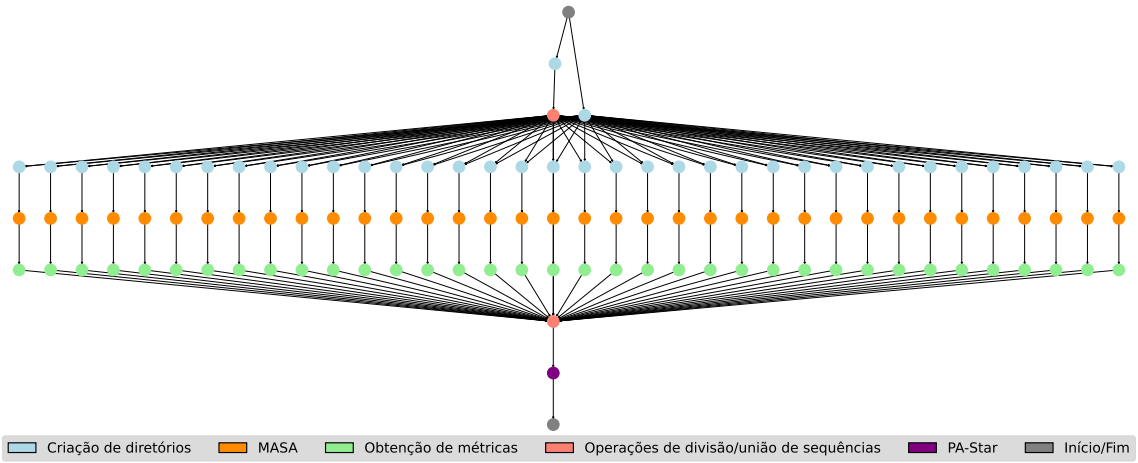


Figura 4. Grafo acíclico direcionado correspondente à execução do *workflow* com 9 sequências. Os nós representam as tarefas, e as arestas indicam as dependências de dados entre elas.

Tabela 1. Desempenho do MASA em execução Sequencial e paralela para diferentes grupos de sequências

Grupos	Quantidade de Pares	Tempo Médio (s) (um par)		Tempo Total (s) (todos os pares)	
		Sequencial	Paralelo	Sequencial	Paralelo
9 seqs	36	0,07	0,23	2,55	8,25
18 seqs	153	0,06	0,09	9,13	14,44
28 seqs	378	0,06	0,11	21,26	40,97
38 seqs	703	0,07	0,11	45,78	74,00

cada uma realizando um alinhamento de forma sequencial, mas permitindo a execução simultânea de diferentes alinhamentos. Para embasar a decisão de usar o *Sequencial* em nosso *workflow*, realizamos experimentos comparativos entre essas duas estratégias. Os resultados estão apresentados na Tabela 1.

A Tabela 1 apresenta, em cada coluna (da esquerda para a direita): os grupos de sequências analisadas, o número de pares gerados a partir da combinação dessas sequências, os tempos médios por par de alinhamento e os tempos médios para o processamento completo de todas as sequências, considerando tanto as versões *Paralela* quanto *Sequencial*, conforme descritas anteriormente. Os resultados indicam que, para o *dataset* em estudo, não é vantajoso executar cada par de sequências em paralelo dentro de uma mesma instância (versão *Paralela*). Em vez disso, é mais eficiente realizar cada alinhamento de forma sequencial (versão *Sequencial*), enquanto se aproveitam os recursos computacionais disponíveis para executar várias instâncias em paralelo. Essa estratégia maximiza a utilização do sistema e reduz o tempo total de execução. Considerando o tempo necessário para alinhar todos os pares, a abordagem *Sequencial* foi $2,11\times$, na média aritmética, mais rápida do que a abordagem *Paralela*.

Fase AMS. Devido à complexidade inerente ao problema de AMS, torna-se computacionalmente inviável executar o PA-Star com um número elevado de sequências. De fato, ao investigarmos a escalabilidade da ferramenta, identificamos que o PA-Star não suporta

mais do que 9 sequências: por aplicar uma solução exata baseada em programão dinâmica, o crescimento do consumo de memória é exponencial, tornando a execução inviável.

De forma geral, as análises apresentadas fundamentam as decisões de implementação adotadas no desenvolvimento do nosso *workflow*. Logo, os resultados reportados na Figura 3 consideram a versão *Sequencial* na Fase AS e o alinhamento múltiplo de 5 sequências na Fase AMS. Com base nesses resultados, observamos que a Fase AS é a principal responsável pelos ganhos de desempenho, sendo capaz de reduzir o tempo total de execução em aproximadamente $77,4\%$. Por outro lado, a Fase AMS tem impacto limitado na performance global, uma vez que não identificamos oportunidades adicionais de otimização – além do paralelismo já explorado internamente pelo próprio PA-Star – que pudessem ser aplicadas de forma eficaz nesta fase.

6. Conclusão

O presente trabalho propõe um *workflow* científico para a realização de AMS exato, explorando de forma eficiente os recursos de PAD disponíveis. Nossos experimentos, considerando a implementação do *workflow* tanto com a biblioteca PyCOMPSs quanto com *scripts Shell*, demonstraram ganhos de $2,08\times$ e $3,95\times$, respectivamente, em comparação com a execução tradicional das etapas do *workflow*. Em particular, a implementação com PyCOMPSs apresentou melhor desempenho à medida que a quantidade de sequências aumentou, alcançando um ganho de $80,8\%$ no alinhamento de 38 sequências. Como trabalhos futuros, pretendemos avaliar outras bases de dados biológicos, contendo um número maior de sequências e/ou sequências de maior comprimento, de forma a investigar mais detalhadamente a escalabilidade da nossa proposta.

7. Agradecimentos

Agradecemos ao Laboratório Nacional de Computação Científica (LNCC) e ao supercomputador Santos Dumont pelo acesso a recursos de PAD. Esta pesquisa foi financiada pelo CNPq (440360/2022-6, 444558/2024-1) e parcialmente financiada pela FAPERJ (E-26/202.302/2024).

Referências

Cohen-Boulakia, S., Belhajjame, K., Collin, O., Chopard, J., Froidevaux, C., Gaignard, A., Hinsen, K., Larmande, P., Bras, Y. L., Lemoine, F., Mareuil, F., Ménager, H., Pradal, C., and Blanchet, C. (2017). Scientific workflows for computational reproducibility in the life sciences: Status, challenges and opportunities. *Future Generation Computer Systems*, 75:284–298.

De O. Sandes, E. F., Miranda, G., Martorell, X., Ayguade, E., Teodoro, G., and De Melo, A. C. (2016). Masa: A multiplatform architecture for sequence aligners with block pruning. *ACM Transactions on Parallel Computing (TOPC)*, 2(4):1–31.

Gao, F., Chen, C., Arab, D. A., Du, Z., He, Y., and Ho, S. Y. W. (2019). EasyCodeML: A visual tool for analysis of selection using CodeML. *Ecology and evolution*, 9(7):3891–3898.

Katoh, K. and Standley, D. M. (2013). Mafft multiple sequence alignment software version 7: improvements in performance and usability. *Molecular biology and evolution*, 30(4):772–780.

Lassmann, T. (2020). Kalign 3: multiple sequence alignment of large datasets. *Bioinformatics (Oxford, England)*, 36.

Needleman, S. B. and Wunsch, C. D. (1970). A general method applicable to the search for similarities in the amino acid sequence of two proteins. *Journal of molecular biology*, 48(3):443–453.

Papadopoulos, J. S. and Agarwala, R. (2007). Cobalt: constraint-based alignment tool for multiple protein sequences. *Bioinformatics*, 23(9):1073–1079.

Prestwich, S., Higgins, D., and O'Sullivan, O. (2003). A sat-based approach to multiple sequence alignment. In *International Conference on Principles and Practice of Constraint Programming*, pages 940–944. Springer.

Schabauer, H., Valle, M., Pacher, C., Stockinger, H., Stamatakis, A., Robinson-Rechavi, M., Yang, Z., and Salamin, N. (2012). SlimCodeML: An Optimized Version of CodeML for the Branch-Site Model. In *2012 IEEE 26th International Parallel and Distributed Processing Symposium Workshops & PhD Forum*, pages 706–714. IEEE.

Sievers, F. and Higgins, D. G. (2018). Clustal omega for making accurate alignments of many protein sequences. *Protein Science*, 27(1):135–145.

Slowinski, J. B. (1998). The number of multiple alignments. *Molecular Phylogenetics and Evolution*, 10(2):264–266.

Smith, T. F., Waterman, M. S., et al. (1981). Identification of common molecular subsequences. *Journal of molecular biology*, 147(1):195–197.

Sundfeld, D., Razzolini, C., Teodoro, G., Boukerche, A., and de Melo, A. C. M. A. (2018). Pa-star: A disk-assisted parallel a-star strategy with locality-sensitive hash for multiple sequence alignment. *Journal of Parallel and Distributed Computing*, 112:154–165. Parallel Optimization using/for Multi and Many-core High Performance Computing.

Suter, F., Coleman, T., Altintaş, İ., Badia, R. M., Balis, B., Chard, K., Colonnelli, I., Deelman, E., Di Tommaso, P., Fahringer, T., et al. (2025). A terminology for scientific workflow systems. *Future Generation Computer Systems*, page 107974.

Tejedor, E., Becerra, Y., Alomar, G., Queralt, A., Badia, R. M., Torres, J., Cortes, T., and Labarta, J. (2017). PyCOMPSs: Parallel computational workflows in Python. *The International Journal of High Performance Computing Applications*, 31(1):66–82.

Tripathi, R., Sharma, P., Chakraborty, P., and Varadwaj, P. K. (2016). Next-generation sequencing revolution through big data analytics. *Frontiers in Life Science*, 9(2):119–149.

Wang, L. and Jiang, T. (1994). On the complexity of multiple sequence alignment. *Journal of computational biology*, 1(4):337–348.

Yang, Z. (2007). PAML 4: Phylogenetic Analysis by Maximum Likelihood. *Molecular Biology and Evolution*, 24(8):1586–1591.

Yoo, A. B., Jette, M. A., and Grondona, M. (2003). SLURM: Simple Linux Utility for Resource Management. In Goos, G., Hartmanis, J., van Leeuwen, J., Feitelson, D., Rudolph, L., and Schwiegelshohn, U., editors, *Job Scheduling Strategies for Parallel Processing*, volume 2862, pages 44–60. Springer Berlin Heidelberg, Berlin, Heidelberg.