
Um Estudo Preliminar sobre Aprendizado por Reforço e Algoritmo Genético para Escalonamento de Jobs

Bernardo Gallo

UFF-Universidade Federal Fluminense
Av. Gal. Milton Tavares de Souza, s/nº São Domingos – Niterói – RJ
bgallo@id.uff.br

Lúcia Drummond

UFF-Universidade Federal Fluminense
Av. Gal. Milton Tavares de Souza, s/nº São Domingos – Niterói – RJ
lucia@ic.uff.br

José Viterbo

UFF-Universidade Federal Fluminense
Av. Gal. Milton Tavares de Souza, s/nº São Domingos – Niterói – RJ
Viterbo @ic.uff.br

RESUMO

Este trabalho apresenta um estudo comparativo entre Aprendizado por Reforço (RL) e Algoritmo Genético (AG) para o problema de escalonamento de *jobs* em supercomputadores. Avaliou-se o desempenho de ambas as técnicas utilizando *logs* de três supercomputadores reais, com foco na métrica de *bounded slowdown* médio. Os resultados demonstram que, para os algoritmos implementados, o AG se destaca por seu baixo custo computacional, obtendo soluções de qualidade em tempo hábil, enquanto o RL alcança um desempenho superior, mas com um custo computacional significativamente maior.

PALAVRAS CHAVE. Escalonamento de Jobs, Aprendizado por Reforço, Algoritmo Genético, Bounded Slowdown.

Tópicos (Escalonamento, Inteligência Artificial, Aprendizado por Reforço e Algoritmo Genético)

ABSTRACT

This paper presents a comparative study between Reinforcement Learning (RL) and Genetic Algorithm (GA) for the job scheduling problem in supercomputers. We evaluate the performance of both techniques using logs from three real supercomputers, focusing on the bounded slowdown metric. The results demonstrate that, for the implemented algorithms, the GA stands out for its low computational cost, obtaining high-quality solutions in a timely manner, while RL achieves superior performance, but at a significantly higher computational cost.

KEYWORDS. Job Scheduling, Reinforcement Learning, Genetic Algorithm, Bounded Slowdown.

Paper topics (Scheduling, Artificial Intelligence, Reinforcement Learning and Genetic Algorithm)

1. Introdução

O escalonamento de *jobs* é um problema de otimização que encontra paralelos em diversas aplicações do mundo real, sendo crucial na área de Computação de Alto Desempenho (HPC). Ele pode ser definido de forma genérica da seguinte maneira [Pinedo, 2016]: Dado um conjunto de n *jobs* $J = \{J_1, J_2, \dots, J_n\}$ e um conjunto de m máquinas $M = \{M_1, M_2, \dots, M_m\}$, o problema de escalonamento consiste em alocar cada *job* a uma ou mais máquinas e determinar a ordem e o tempo de processamento de cada *job*, de forma a otimizar um ou mais objetivos.

O objetivo do problema de escalonamento é encontrar uma atribuição de *jobs* a máquinas ao longo do tempo, que minimize ou maximize uma função objetivo específica, como: Tempo de conclusão máximo, atraso máximo ou *bounded slowdown*.

Devido à sua natureza combinatória e complexidade computacional, este problema é classificado como NP-difícil [Garey et al., 1976], sendo amplamente estudado em contextos como manufatura, computação em nuvem e sistemas embarcados.

Tradicionalmente, heurísticas têm sido utilizadas para abordar o escalonamento de *jobs* [Hajariwala et al., 2024]. Algoritmos Genéticos AGs [Goldberg, 1989], inspirados nos princípios da evolução natural, têm sido amplamente aplicados em problemas de escalonamento devido à sua capacidade de explorar eficientemente grandes espaços de busca. Pesquisas indicam que AGs podem produzir soluções de alta qualidade para o escalonamento de *jobs*, especialmente quando combinados com outras técnicas heurísticas.

No entanto, com o aumento da complexidade e dinamicidade das cargas de trabalho modernas, essas abordagens podem mostrar-se limitadas, pois heurísticas tradicionais e meta-heurísticas, apesar de eficientes, muitas vezes não conseguem adaptar-se em tempo real a mudanças frequentes nas características dos jobs ou a contextos altamente dinâmicos, como é o caso de sistemas HPC modernos com heterogeneidade de recursos e variações temporais nos workloads. Nesse cenário, técnicas de inteligência artificial, como o Aprendizado de máquina (ML), emergem como alternativas promissoras. Uma dessas técnicas, aprendizado por reforço (RL), tem sido mais recentemente utilizada para resolver esse problema. RL permite que os agentes aprendam políticas de decisão por meio da interação com o ambiente, maximizando uma função de recompensa. Estudos recentes demonstram a eficácia de abordagens baseadas em RL para problemas de escalonamento [Bonetta et al., 2023], destacando melhorias significativas em métricas como *throughput*, *bounded slowdown*, *makespan* e tempo médio de espera.

Apesar dos avanços de cada uma das abordagens, comparações sistemáticas entre ML e Metaheurísticas em contextos realistas de escalonamento de *jobs* ainda são escassas. Este trabalho propõe um estudo comparativo entre ML e meta-heurísticas, DeepRM e AG, avaliando sua eficácia em uma métrica de desempenho, o *bounded slowdown* médio. O objetivo é identificar vantagens, limitações e condições ideais para o uso de cada técnica, contribuindo com evidências práticas para a escolha de métodos inteligentes no contexto de escalonamento. Utilizou-se *logs* de três supercomputadores reais, aplicando suas configurações públicas em ambos os algoritmos para assegurar a imparcialidade dos experimentos.

O restante deste artigo está estruturado da seguinte forma: A Seção 2 apresenta uma revisão dos trabalhos relacionados, abordando o uso de Aprendizado por Reforço e Algoritmos Genéticos no contexto do escalonamento de jobs. A Seção 3 define formalmente o problema de escalonamento de jobs investigado neste estudo, incluindo as características dos jobs e das máquinas, bem como a métrica de avaliação utilizada. A Seção 5 apresenta os resultados experimentais, comparando o desempenho dos algoritmos em diferentes cenários. Por fim, a Seção 6 sumariza as principais contribuições e discute possíveis direções para trabalhos futuros.

2. Trabalhos Relacionados

Nesta seção, apresenta-se uma revisão de trabalhos recentes e relevantes que aplicam Aprendizado por Reforço (RL) e Algoritmos Genéticos (AG) ao problema de escalonamento de *jobs*. Dada a importância do escalonamento de *jobs* em Pesquisa Operacional, especialmente em contextos complexos como a Computação de Alto Desempenho (HPC) e a computação em nuvem, focam-se em artigos que exploram essas técnicas para otimizar a alocação de recursos e o escalonamento de *jobs* nesses contextos.

2.1. Aprendizado por Reforço para escalonamento

O RL tem se destacado como uma abordagem eficaz para o escalonamento de *jobs*, especialmente em ambientes dinâmicos como a computação em nuvem. Gu et al. [2025] apresenta uma análise abrangente das metodologias de RL para otimizar o escalonamento de *jobs* e o gerenciamento de recursos em ambientes de computação em nuvem. O foco principal é a revisão de estudos existentes, categorizados pelos algoritmos de RL empregados. O artigo explora a utilização de processos de decisão de Markov para modelar problemas de otimização nessas áreas, demonstrando o potencial do RL para abordá-los eficazmente. Da mesma forma, o artigo oferece uma visão geral dos algoritmos RL contemporâneos e examina suas aplicações no escalonamento de *jobs*, incluindo o escalonamento de *jobs* e de *workflows*, bem como no gerenciamento de recursos, com ênfase no provisionamento e no escalonamento de recursos. Esse artigo fornece um panorama atual e sistemático das principais técnicas de RL utilizadas, além de discutir os desafios e tendências da área, sendo uma excelente base teórica para a motivação do uso de RL no presente trabalho.

Além disso, Tassel et al. [2021] desenvolveram um ambiente de RL para o problema de escalonamento de *jobs*, demonstrando que abordagens baseadas em RL podem superar métodos tradicionais em *benchmarks* clássicos. Os autores utilizaram um método ator-crítico com apenas um agente e *Proximal Policy Optimization* (PPO) para tal agente aprender a escalonar os *jobs* com a experiência. O trabalho de Tassel et al. apresenta um ambiente experimental bem definido para escalonamento com RL, utilizando algoritmos modernos. Sua relevância se dá por mostrar que agentes de RL podem superar heurísticas clássicas, o que se alinha diretamente com os objetivos deste estudo comparativo.

Em Mao et al. [2016] os autores apresentam o DeepRM, um sistema que utiliza RL para o escalonamento de *jobs* em clusters de recursos múltiplos. Os autores demonstram que o DeepRM é capaz de aprender políticas de alocação de recursos eficientes, alcançando desempenho comparável ou superior a heurísticas tradicionais. Este artigo fornece a base para a implementação do algoritmo de RL utilizado no presente trabalho, especialmente no que tange à formulação do problema de escalonamento como um problema de RL e o emprego de redes neurais profundas para aproximar a política de decisão.

2.2. Algoritmos Genéticos para escalonamento

Os AGs têm sido aplicados com sucesso ao problema de escalonamento de *jobs*, devido à sua capacidade de explorar grandes espaços de busca e encontrar soluções próximas do ótimo. Gen e Cheng [1999] apresentaram um estudo sobre o uso de AGs em problemas de escalonamento de *jobs*, discutindo diferentes representações e estratégias híbridas.

Particularmente, Arroyo e Vianna [2004] aborda o problema de escalonamento de *jobs* em máquinas paralelas idênticas com tempos de preparação dependentes da sequência, visando minimizar simultaneamente dois critérios: o *makespan* e o atraso total em relação às datas de entrega. Os autores propõem três algoritmos genéticos diferentes, cada um incorporando estratégias de busca local, para encontrar soluções aproximadas eficientes. Os resultados demonstram que os algoritmos propostos apresentam bom desempenho tanto em termos de qualidade das soluções quanto no tempo de execução. Como Arroyo e Vianna [2004] tiveram sucesso ao usar um AG com busca local

para solucionar o problema de escalonamento com múltiplos critérios de otimização, este artigo inspirou a implementação do AG formulado no presente trabalho.

Paes e Pessoa [2016] apresenta um algoritmo genético com inserção gulosa para resolver o problema de escalonamento *job shop* sem espera, visando minimizar o *makespan*. Os autores propõem uma abordagem eficiente que insere os *jobs* de forma sequencial, conforme a ordem no cromossomo, sem sobreposição com os demais *jobs* já inseridos. Os resultados demonstram desempenho superior da abordagem proposta em instâncias de grande porte, comparado com outras técnicas da literatura. O artigo foi destacado aqui por apresentar uma adaptação prática e eficaz de AG para o problema *job shop*. A técnica de inserção gulosa utilizada fornece uma perspectiva alternativa e relevante à abordagem adotada neste estudo, especialmente no contexto de instâncias reais e de trabalhos futuros.

3. Modelagem do Problema

3.1. Definição do Problema de Escalonamento

O problema de escalonamento de *jobs* abordado neste trabalho é formalmente definido da seguinte forma:

- Um conjunto de N *jobs* $J = \{J_1, J_2, \dots, J_n\}$, cada um caracterizado por:
 - Tempo de submissão ($submitTime_i$): O momento em que o *job* se torna disponível para processamento.
 - Tempo de execução ($runTime_i$): O tempo de processamento necessário para *job*.
 - Número de CPUs necessárias ($numCPUs_i$): O número de unidades de processamento necessárias.
 - Requisito de memória ($memReq_i$): A quantidade de memória necessária.
- Um conjunto de m máquinas $M = \{M_1, M_2, \dots, M_m\}$, cada uma caracterizada por:
 - Total de CPUs ($totalCPUs_k$): O número total de unidades de processamento disponíveis.
 - Total de RAM ($totalRAM_k$): A quantidade total de memória disponível.

No contexto de escalonamento de *jobs*, o conceito de *slowdown* é usado para medir o impacto do tempo de espera na conclusão de um *jobs*. O *slowdown* de um *job* é definido como a razão entre o tempo gasto para completar o *job* (tempo de espera + tempo de execução) e o tempo de execução real do *jobs*. Formalmente, o *slowdown* (S_i) de um *job* J_i é dado por:

$$S_i = \frac{waitTime_i + runTime_i}{runTime_i}$$

onde $waitTime_i$ é o tempo que o *job* J_i espera antes de iniciar a execução e $runTime_i$ é o tempo de execução do *job*.

No entanto, o uso direto do *slowdown* pode levar a uma avaliação distorcida do desempenho do escalonamento, especialmente quando há muitos *jobs* curtos. *Jobs* com tempos de execução muito pequenos podem apresentar valores de *slowdown* desproporcionalmente altos, mesmo com pequenos tempos de espera, afetando negativamente a média geral do desempenho do escalonador.

Para mitigar esse problema, a métrica do *bounded slowdown* médio é frequentemente empregada. Essa métrica limita o impacto de *jobs* curtos no cálculo do *slowdown*. O *bounded slowdown* (BS_i) de um *job* J_i é calculado como:

$$BS_i = \frac{waitTime_i + runTime_i}{\max(runTime_i, threshold)}$$

onde *threshold* é um valor predefinido que representa um tempo de execução mínimo considerado significativo. Ao usar o máximo entre o tempo de execução real e o limite (*threshold*) no denominador, o impacto de *jobs* com tempo de execução menor que o limite é reduzido.

O *bounded slowdown* médio do escalonamento é então calculado como a média dos *bounded slowdowns* de todos os *jobs* escalonados:

$$\overline{BS} = \frac{1}{n} \sum_{i=1}^n BS_i$$

3.2. Implementação do Algoritmo Genético (AG)

O AG implementado segue a estrutura canônica delineada por Goldberg e Mitchell, com adaptações específicas para o contexto de escalonamento de *jobs* [Goldberg, 1989; Hajariwala et al., 2024].

3.2.1. Codificação do Cromossomo

Um cromossomo representa uma solução potencial para o problema de escalonamento. Nesta implementação, um cromossomo é codificado como um vetor de estruturas ‘JobAssignment’. Cada ‘JobAssignment’ mapeia um *job* para uma ou mais máquinas e atribui a ele uma prioridade:

- ‘jobId’: Identificador do *job*.
- ‘machineIds’: Um vetor de identificadores de máquinas às quais o *job* é atribuído. Isso permite que os *jobs* sejam processados em várias máquinas, se os recursos estiverem disponíveis.
- ‘priority’: Um inteiro que indica a prioridade de escalonamento do *job*.

Essa codificação oferece flexibilidade na atribuição de *jobs* e na priorização de sua execução.

Inicialização

A população inicial de cromossomos é gerada usando uma combinação de estratégias para garantir a diversidade:

- **Inicialização Sensível a Múltiplas Máquinas:** Um subconjunto da população é criado considerando os requisitos de recursos dos *jobs* e os recursos disponíveis das máquinas, tentando alocar *jobs* em várias máquinas quando benéfico.
- **Inicialização Inspirada no FCFS:** Outro subconjunto imita a política de escalonamento First-Come-First-Served (FCFS), atribuindo *jobs* na ordem de seus tempos de submissão, respeitando as restrições de recursos.
- **Inicialização Aleatória:** Os cromossomos restantes são gerados atribuindo *jobs* aleatoriamente às máquinas para introduzir maior variabilidade na população.

A lógica por trás dessa inicialização mista é semear o AG com soluções informadas e exploratórias, seguindo o princípio de que uma população inicial diversificada pode melhorar a capacidade do algoritmo de escapar de ótimos locais.

3.2.2. Função de Fitness

A função de *fitness* avalia a qualidade de um determinado escalonamento (representado por um cromossomo). O escalonamento é simulado para determinar os tempos de início e fim dos *jobs*, permitindo o cálculo de métricas relevantes. A função de *fitness* combina vários fatores:

- **Slowdown Médio Limitado:** Slowdown é definido como a razão entre o tempo de conclusão de um *job* e seu tempo de execução. O *bounded slowdown* mitiga o impacto de *jobs* muito curtos na média. Esta métrica é minimizada, portanto utiliza-se o seu negativo no cálculo.
- **Número de *Jobs* Escalonadas:** Um bônus é concedido para escalonar mais *jobs*, incentivando o AG a encontrar soluções que maximizem a vazão.

A função de *fitness* atribui pesos a esses componentes para equilibrar os objetivos concorrentes.

3.2.3. Seleção

A seleção é o processo de escolher cromossomos pais para reprodução. O AG implementado emprega a seleção por torneio, um método comum e eficiente [Miller e Goldberg, 1996].

- **Seleção por Torneio:** Um subconjunto da população (tamanho do torneio) é selecionado aleatoriamente. O cromossomo com a maior *fitness* dentro desse subconjunto é escolhido como pai. Esse processo é repetido para selecionar vários pais.

Para aprimorar ainda mais o processo de seleção, um mecanismo de reconhecimento de diversidade é incorporado:

- **Seleção por Torneio com Reconhecimento de Diversidade:** Quando a melhor solução não se altera por dez gerações, indicando potencial convergência para um ótimo local, o processo de seleção é modificado para favorecer cromossomos mais diferentes dos outros na população. Isso é alcançado adicionando uma pontuação de diversidade à pontuação de *fitness*, ponderada por um parâmetro ‘*diversityWeight*’. A pontuação de diversidade é calculada com base nas diferenças das atribuições de máquinas entre os cromossomos.

3.2.4. Crossover

O *crossover* é o operador genético que combina o material genético de dois cromossomos pais para criar descendentes. O AG implementado utiliza um *two-point crossover* modificado:

- **Two-point crossover** Dois pontos são selecionados aleatoriamente ao longo da sequência de *jobs*. O cromossomo filho herda as atribuições de um pai entre esses pontos e do outro pai fora desses pontos.

Essa estratégia de *crossover* visa explorar boas características de ambos os pais, ao mesmo tempo em que introduz variabilidade.

3.2.5. Mutação

A mutação introduz mudanças aleatórias nos cromossomos para manter a diversidade e explorar novas regiões do espaço de soluções. O AG implementado utiliza uma estratégia de mutação adaptativa:

-
- **Mutação Adaptativa:** A taxa de mutação, que controla a probabilidade de um gene (atribuição de máquina) ser alterado, é ajustada dinamicamente durante o processo de busca. Se a diversidade da população for baixa ou a melhoria estagnar, a taxa de mutação é aumentada para incentivar a exploração. Inversamente, se houver muitas mutações, a taxa é diminuída para promover a exploração de soluções promissoras. A diversidade da população é medida com base na distância de Hamming entre cromossomos (levando em conta as atribuições de máquina e prioridade).
 - **Operadores de Mutação:** Vários operadores de mutação são empregados, incluindo:
 - Adicionar uma máquina à atribuição de um *job*.
 - Remover uma máquina da atribuição um *job*.
 - Substituir uma máquina em uma atribuição de um *job*.

Esses operadores visam perturbar o escalonamento de várias maneiras, potencialmente descobrindo melhores soluções.

3.2.6. Busca Local

Para refinar ainda mais as soluções obtidas pelo AG, um procedimento de busca local é aplicado. O AG implementado incorpora uma estratégia de busca de balanceamento de carga:

- **Busca de Balanceamento de Carga:** Essa estratégia tenta melhorar o escalonamento movendo *jobs* de máquinas fortemente carregadas para máquinas levemente carregadas, otimizando assim a utilização dos recursos.

A busca local ajuda a ajustar as soluções e, potencialmente, escapar de ótimos locais nos quais o AG pode ficar preso.

3.2.7. Critérios de Término

A execução do AG termina com base nos seguintes critérios:

- **Número Máximo de Gerações:** O algoritmo é executado por um número predefinido de gerações.
- **Convergência:** Se o *fitness* da melhor solução não melhorar por um certo número de gerações, o algoritmo termina, assumindo que convergiu para um ótimo local.

Esses critérios equilibram a necessidade de explorar o espaço de soluções de forma completa com o desejo de obter uma solução em um tempo razoável.

Os experimentos com o Algoritmo Genético foram realizados com os seguintes parâmetros, escolhidos com base em estudos prévios da literatura e ajustes empíricos:

- **Tamanho da população** (*POPULATION_SIZE*): 200 cromossomos.
- **Número máximo de gerações** (*NUM_GENERATIONS*): 100.
- **Taxa de mutação** (*MUTATION_RATE*): 0,5.
- **Tamanho do torneio de seleção** (*TOURNAMENT_SIZE*): 200.
- **Número máximo de gerações sem melhoria** (*THRESHOLD*): 10.
- **Peso da diversidade na seleção adaptativa** (*DIVERSITY_WEIGHT*): 0,3.

3.3. Modelagem do Aprendizado de Máquina

O RL constitui um paradigma do aprendizado de máquina que se concentra na tomada de decisão sequencial por um agente autônomo em um ambiente dinâmico. Diferentemente do aprendizado supervisionado, que requer pares de entrada-saída rotulados, o AR permite que o agente aprenda através da interação direta com o ambiente, recebendo recompensas ou penalidades como *feedback* para suas ações. O objetivo fundamental no RL é que o agente aprenda uma política ótima, que define o comportamento do agente ao mapear estados do ambiente para ações, de forma a maximizar a recompensa cumulativa esperada ao longo do tempo.

Formalmente, o processo de RL pode ser descrito como um Processo de Decisão de Markov, definido por uma tupla (S, A, P, R, γ) , onde:

- S representa o conjunto de todos os possíveis **estados** do ambiente.
- A denota o conjunto de todas as possíveis **ações** que o agente pode realizar.
- $P(s'|s, a)$ é a probabilidade de transição do ambiente do estado s para o estado s' após o agente executar a ação a .
- $R(s, a, s')$ é a **recompensa** recebida pelo agente após a transição do estado s para s' devido à ação a .
- $\gamma \in [0, 1]$ é o **fator de desconto**, que determina a importância das recompensas futuras em relação às recompensas imediatas.

A **política** do agente, denotada por $\pi(a|s)$, especifica a probabilidade de selecionar a ação a no estado s . O objetivo do AR é encontrar a política π^* que maximiza a função de valor, que representa a recompensa cumulativa descontada esperada.

3.3.1. Policy Gradient

Os métodos de *Policy Gradient* constituem uma classe de algoritmos de RL que otimizam a política diretamente através da realização de uma descida do gradiente no espaço de parâmetros da política. O gradiente do objetivo, que é a recompensa cumulativa descontada esperada, é dado por:

$$\nabla_{\theta} \mathbb{E} \left[\sum_{t=0}^{\infty} \gamma^t r_t \right] = \mathbb{E}_{\pi_{\theta}} \left[\sum_{t=0}^{\infty} \nabla_{\theta} \log \pi_{\theta}(s_t, a_t) Q^{\pi_{\theta}}(s_t, a_t) \right]$$

onde:

- θ representa os parâmetros da política.
- $\pi_{\theta}(a|s)$ denota a política parametrizada por θ .
- $Q^{\pi_{\theta}}(s_t, a_t)$ é a função de ação-valor, representando a recompensa cumulativa descontada esperada ao tomar a ação a_t no estado s_t e seguir a política π_{θ} subsequentemente.

A atualização dos parâmetros da política é realizada na direção do gradiente:

$$\theta \leftarrow \theta + \alpha \nabla_{\theta} \log \pi_{\theta}(s_t, a_t) v_t$$

onde α é a taxa de aprendizado e v_t é uma estimativa da recompensa cumulativa obtida a partir de uma trajetória de execução.

Para mitigar a alta variância inerente às estimativas de gradiente, uma linha de base b_t é subtraída da recompensa:

$$\theta \leftarrow \theta + \alpha \nabla_{\theta} \log \pi_{\theta}(s_t, a_t)(v_t - b_t)$$

3.3.2. Formulação do Problema de Escalonamento com DeepRM

No contexto do DeepRM[Mao et al., 2016], o problema de escalonamento de *jobs* em um *cluster* de recursos múltiplos é formulado como um problema de RL da seguinte maneira:

- **Estado:** O estado do sistema é representado por uma combinação de imagens que capturam a alocação de recursos do *cluster* e os requisitos de recursos dos *jobs* aguardando na fila. Adicionalmente, uma segunda fila contém a informação de *jobs* além dos primeiros M *jobs*.
- **Ações:** O espaço de ações compreende a decisão de agendar qualquer um dos M *jobs* pendentes ou tomar uma ação nula ($\emptyset$) para indicar a ausência de escalonamento.
- **Recompensas:** A função de recompensa é projetada para refletir o objetivo de otimização do sistema. Para minimizar o slowdown médio do *job*, a recompensa r_t no timestep t é definida como:

$$r_t = \sum_{j \in \mathcal{J}} \frac{-1}{T_j}$$

onde $\mathcal{J}$ representa o conjunto de *jobs* atualmente no sistema e T_j é a duração do *job* j .

O DeepRM emprega uma rede neural profunda (DNN) para aproximar a política, permitindo que o agente aprenda estratégias de escalonamento eficazes a partir da experiência.

4. Materiais e métodos

Para a avaliação comparativa entre o AG e o RL utilizou-se três *logs* de supercomputadores. Os dados foram estruturados no formato Standard Workload Format (SWF), um padrão amplamente adotado para a representação das workloads em sistemas de computação paralela. O SWF define um conjunto de campos para descrever as características dos *jobs*, incluindo, mas não se limitando a: tempo de submissão, tempo de execução, número de unidades de processamento requisitadas, quantidade de memória alocada e identificadores de usuário e grupo. A utilização deste formato padronizado facilita a reprodutibilidade dos experimentos e a comparação com outros estudos na área. O primeiro deles foi o KIT-FH2-2016-1 do *Karlsruhe Institute of Technology ForHLR II System* contendo 114355 *jobs* executados de 2016 a 2018. O segundo deles foi o log do Polaris, um supercomputador do *Argonne Leadership Computing Facility*, que é uma instalação de usuário do Escritório de Ciência do DOE apoiada sob o Contrato DE-AC02-06CH11357, que contém 286958 *jobs* executados no ano de 2024. Por último temos o Theta, um supercomputador do *Argonne Leadership Computing Facility*, que é uma instalação de usuário do Escritório de Ciência do DOE apoiada sob o Contrato DE-AC02-06CH11357, que contém 113 *jobs* executados no ano de 2024. Cada *log* apresenta características distintas em relação ao volume de *jobs*, variedade dos requisitos e dinamicidade do sistema, proporcionando um cenário abrangente de avaliação. Para o treinamento do modelo de RL foi separado 80% dos *jobs* de cada *log*, todos os resultados experimentais apresentados foram testes com os últimos 20% dos respectivos *logs*, os modelos foram treinados por 1000 episódios cada.

Tabela 1: Bounded slowdown médio para diferentes algoritmos

Algoritmo	Polaris	KIT-FH2	Theta
DeepRM	1.94	2.19	2.98
FCFS	31.60	4.93	32.59
SJF	29.30	2.86	30.28
AG	18.90	2.44	17.29

5. Resultados Experimentais

Como pode-se perceber baseado na Tabela 1 os resultados obtidos demonstram que algoritmo de RL DeepRM atingiu 1.94 de *bounded slowdown* médio nos dados obtidos do Polaris com enquanto o AG atingiu 18.9, já nos dados KIT-FH2-2016-1 o DeepRM atingiu 2.19 contra 2.44, usando os *jobs* do Theta o DeepRM atingiu 2.98 de *bounded slowdown* médio enquanto o AG atingiu 17.29. Tanto o AG quanto o DeepRM sempre superaram heurísticas clássicas como *Shortest Job First*(SJF) e *First Come First Serve*(FCFS) nos testes realizados. O FCFS é uma política de escalonamento que aloca recursos aos jobs na ordem em que eles chegam na fila de espera, ou seja, o primeiro job a ser submetido é o primeiro a ser executado [Grosz et al., 2021]. O SJF prioriza os *jobs* mais curtos com o objetivo de otimizar o *throughput* do sistema, contudo essa heurística sofre de *starvation*, quando é possível um *job* esperar tempo indeterminado na fila. [Jikui et al., 2010]

O algoritmo DeepRM possui um custo computacional significativamente maior para o *Dataset*, o treinamento levou cerca de 24 horas em um computador com um Intel Core i9-13900 com 32 cores, 62 GB de memória e uma GPU NVIDIA A5000, enquanto o AG levou menos de três minutos para escalonar a mesma instância de problema. Como um algoritmo de aprendizado de máquina, a maior parte desse custo é em tempo de treinamento, não de inferência.

A natureza do DeepRM como um algoritmo de aprendizado de máquina implica que a maior parte do custo computacional está associada à fase de treinamento do modelo. Uma vez treinado, o modelo pode realizar inferências (ou seja, escalonar novos jobs) de forma relativamente rápida. No entanto, o tempo necessário para treinar o modelo inicialmente pode ser proibitivo em cenários onde a agilidade e a adaptabilidade são essenciais.

Em contraste, a eficiência computacional do AG o torna uma alternativa atraente para aplicações práticas que demandam respostas rápidas e otimização de recursos. Sua capacidade de fornecer soluções de alta qualidade em um tempo computacionalmente inferior ao do RL o torna adequado para contextos onde o tempo de resposta é crítico ou os recursos computacionais são limitados.

6. Conclusão

Em síntese, este estudo comparativo entre RL e AG para o problema de escalonamento de jobs em supercomputadores revelou um interessante trade-off entre desempenho e custo computacional. Os resultados experimentais demonstraram que o algoritmo de RL, DeepRM, alcança um desempenho superior em termos de *bounded slowdown* médio em comparação com o AG. No entanto, essa melhoria é obtida à custa de um custo computacional significativamente maior, o que pode limitar sua aplicabilidade em cenários com restrições de tempo ou recursos computacionais. Em contrapartida, o AG apresenta um desempenho competitivo, superando heurísticas clássicas e oferecendo soluções de alta qualidade em um tempo computacionalmente viável. Essa eficiência torna o AG uma alternativa atraente para contextos onde a agilidade e a otimização de recursos são cruciais.

Adicionalmente, observou-se que as características dos workloads dos supercomputadores exercem influência sobre o desempenho relativo dos algoritmos. Workloads com maior varia-

bilidade nos tamanhos dos jobs e padrões de chegada mais dinâmicos podem acentuar o desempenho superior do RL em termos de otimização do bounded slowdown, enquanto o AG mantém sua eficiência computacional em diferentes cenários. A capacidade de generalização do RL para diferentes ambientes de escalonamento e workloads foi evidenciada nos experimentos, demonstrando seu potencial para se adaptar a condições variadas.

Em termos de implicações para a prática, os resultados sugerem que a escolha entre RL e AG deve considerar as prioridades do contexto de aplicação. Em cenários onde a otimização do bounded slowdown é primordial e o custo computacional é menos restritivo, o RL surge como uma opção promissora. Por outro lado, em situações que exigem agilidade e eficiência computacional, o AG se apresenta como uma alternativa eficaz.

6.1. Limitações e Direções Futuras

Nesse estudo, enquanto o DeepRM exigiu cerca de 24 horas de treinamento em uma configuração de hardware robusta, o AG foi capaz de escalonar a mesma instância do problema em menos de três minutos. Essa diferença ressalta a eficiência computacional do AG, tornando-o uma alternativa atraente em cenários onde os recursos computacionais são limitados ou o tempo de resposta é crítico. No entanto, é importante destacar as limitações do presente estudo. O escalonamento é um problema clássico com diversas soluções propostas para suas diferentes formulações. Esse artigo, entretanto, limitou-se à implementação de um dos primeiros algoritmos de RL da literatura [Mao et al., 2016] e um algoritmo genético de própria elaboração, com base no algoritmo clássico proposto por Goldberg e Mitchell [Goldberg, 1989]. É possível que outras metaheurísticas tragam resultados superiores ao AG aqui estudado, ou que adaptações do mesmo gerem resultados significativamente melhores. Embora o DeepRM apresente resultados promissores, é importante reconhecer suas limitações e considerar direções futuras para aprimorá-lo:

- **Fronteiras de Máquina e Localidade:** A formulação do problema no DeepRM assume um único "grande pool de recursos", abstraindo as fronteiras da máquina e a potencial fragmentação de recursos. Embora essa formulação seja prática em cenários onde os escalonamentos de *Cluster* tomam decisões independentes por máquina, ela não leva em consideração as questões de localidade de dados, que são importantes para o desempenho.
- **Codificação do Espaço e Escalabilidade** Como o DeepRM modela o estado do sistema e das filas como imagens, seu custo computacional, tanto em tempo quanto em memória, é significativo e ele sofre com uma péssima escalabilidade.

Ao abordar essas limitações, pesquisas futuras podem aprimorar ainda mais a aplicabilidade de ambas as abordagens a cenários de escalonamento do mundo real e avançar no estado da arte no escalonamento, em especial no âmbito de Computação de Alto Desempenho.

Referências

- Arroyo, J. E. C. e Vianna, D. S. (2004). Algoritmos genéticos para o problema de programação de tarefas em máquinas paralelas idênticas com dois critérios. In *Anais do XXXVI Simpósio Brasileiro de Pesquisa Operacional (SBPO)*, p. 1538–1547, João Pessoa, PB, Brasil. Sociedade Brasileira de Pesquisa Operacional. URL https://www.dcc.fc.up.pt/~ines/enia07_html/pdf/28100.pdf.
- Bonetta, G., Zago, D., Cancelliere, R., e Grosso, A. (2023). Job shop scheduling via deep reinforcement learning: a sequence to sequence approach. URL <https://arxiv.org/abs/2308.01797>.

-
- Garey, M. R., Johnson, D. S., e Sethi, R. (1976). The complexity of flowshop and jobshop scheduling. *Mathematics of Operations Research*, 1(2):117–129. URL <https://doi.org/10.1287/moor.1.2.117>.
- Gen, M. e Cheng, R. (1999). *Scheduling Problems*, chapter 6, p. 235–296. John Wiley Sons, Ltd. ISBN 9780470172261. URL <https://onlinelibrary.wiley.com/doi/abs/10.1002/9780470172261.ch6>.
- Goldberg, D. E. (1989). *Genetic Algorithms in Search, Optimization and Machine Learning*. Addison-Wesley Longman Publishing Co., Inc., USA, 1st edition. ISBN 0201157675.
- Grosof, I., Yang, K., Scully, Z., e Harchol-Balter, M. (2021). Nudge: Stochastically improving upon fcfs. *PROCEEDINGS OF THE ACM ON MEASUREMENT AND ANALYSIS OF COMPUTING SYSTEMS*, 5(2). ACM SIGMETRICS Conference, ELECTR NETWORK, 2021.
- Gu, Y., Liu, Z., Dai, S., Liu, C., Wang, Y., Wang, S., Theodoropoulos, G., e Cheng, L. (2025). Deep reinforcement learning for job scheduling and resource management in cloud computing: An algorithm-level review. URL <https://arxiv.org/abs/2501.01007>.
- Hajariwala, D. C., Patil, S. S., e Patil, S. M. (2024). A review of metaheuristic algorithms for job shop scheduling. *Engineering Access*, 11(1):65–91. URL <https://ph02.tci-thaijo.org/index.php/mijet/article/view/254023>.
- Jikui, W., Junrong, L., e Huiyong, W. (2010). A new algorithm to prove that the short-job-first algorithm has the least mean waiting time. In Ma, P. e Qaiser, M., editors, *2ND INTERNATIONAL SYMPOSIUM ON COMPUTER NETWORK AND MULTIMEDIA TECHNOLOGY (CNMT 2010)*, VOLS 1 AND 2, p. 734–735, 345 E 47TH ST, NEW YORK, NY 10017 USA. Huazhong Normal Univ; Huazhong Univ Sci & Technol; Wuhan Univ; Natl Technol Univ Ukraine; Royal Inst Technol; Columbia Univ; Harbin Univ Technol; Res Assoc Modern Educ & Comp Sci; IEEE Educ Soc; IEEE Wuhan Sect E Chapter; IEEE, IEEE. ISBN 978-1-4244-8069-2. 2nd International Symposium on Computer Network and Multimedia Technology (CNMT), Wuhan, PEOPLES R CHINA, DEC 24-26, 2010.
- Mao, H., Alizadeh, M., Menache, I., e Kandula, S. (2016). Resource management with deep reinforcement learning. In *Proceedings of the 15th ACM Workshop on Hot Topics in Networks*, HotNets '16, p. 50–56, New York, NY, USA. Association for Computing Machinery. ISBN 9781450346610. URL <https://doi.org/10.1145/3005745.3005750>.
- Miller, B. L. e Goldberg, D. E. (1996). Genetic algorithms, selection schemes, and the varying effects of noise. *Evolutionary Computation*, 4(2):113–131. ISSN 1063-6560. URL <https://doi.org/10.1162/evco.1996.4.2.113>.
- Paes, F. G. e Pessoa, A. A. (2016). Um algoritmo genético com inserção gulosa para o problema de escalonamento de tarefas job shop sem espera. In *Anais do XLVIII Simpósio Brasileiro de Pesquisa Operacional (SBPO)*, p. 2048–2059, Vitória, ES, Brasil. Sociedade Brasileira de Pesquisa Operacional. URL <https://www.din.uem.br/sbpo/sbpo2016/pdf/156670.pdf>.
- Pinedo, M. L. (2016). *Scheduling: Theory, Algorithms, and Systems*. Springer.
- Tassel, P., Gebser, M., e Schekotihin, K. (2021). A reinforcement learning environment for job-shop scheduling. URL <https://arxiv.org/abs/2104.03760>.