Job Shop Scheduling

Henrique Andrade Lopes[1]

[1]DPI – Universidade Federal de Viçosa (UFV)
Campus Universitário - 36570-900, Viçosa, MG – Brasil

{henrique.a.lopes@ufv.br}

Abstract. *This report presents the implementation and results achieved by a genetic heuristic to solve the production scheduling problem known as Job Shop Scheduling (JSS). JSS is a complex challenge that involves the efficient allocation of tasks to machines, considering precedence constraints and variable processing times. The obtained results demonstrate the effectiveness of the genetic heuristic in minimizing the total production time and complying with the constraints, generating a solution considered good within the specified time frame to solve it. The comparison with existing methods also highlights the advantages of the proposed approach, showcasing its applicability and potential to handle complex scheduling problems in practice.*

Resumo. *Este relatório apresenta a implementação e os resultados alcançados por uma heurística genética para resolver o problema de escalonamento de produção conhecido como Job Shop Scheduling (JSS). O JSS é um desafio complexo que envolve a alocação eficiente de tarefas em máquinas, considerando restrições de precedência e tempos de processamento variáveis. Os resultados obtidos demonstram a eficácia da heurística genética na minimização do tempo total de produção e no cumprimento das restrições, gerando uma solução considerada boa dentro do tempo estipulado para resolvê-la.*

1. Introdução

O problema de Job Shop Scheduling (JSP)[3] é um desafio fundamental na área de otimização de produção, que visa encontrar um agendamento eficiente para a execução de um conjunto de tarefas em máquinas. O objetivo principal é minimizar o tempo total de produção, levando em consideração as restrições de precedência e os tempos de processamento variáveis.

O JSP é conhecido por sua complexidade, pois envolve a alocação adequada de tarefas em máquinas, levando em conta as dependências entre as tarefas e as restrições de precedência. Cada tarefa pertence a um determinado job e precisa ser processada em uma sequência específica, respeitando as restrições impostas pelo problema.

Neste contexto, a heurística genética tem sido amplamente utilizada como uma abordagem eficiente para resolver o problema de escalonamento de produção do JSP. Essa abordagem combina técnicas de algoritmos genéticos com a busca de soluções em um espaço de busca complexo. A heurística genética busca encontrar soluções de alta qualidade em um tempo razoável, explorando o conceito de seleção natural e evolução das soluções.

Este relatório apresenta a implementação e os resultados alcançados por uma heurística genética no problema de Job Shop Scheduling. Serão discutidos os detalhes da implementação, bem como os resultados obtidos em termos de minimização do tempo total de produção e cumprimento das restrições.

1.1. Funcionamento do algoritmo

O algoritmo implementado no arquivo main.cpp espera receber as instâncias no formato fornecido pelo JSPLIB[4], que consiste em uma representação das tarefas e máquinas necessárias para o agendamento. Junto com o trabalho, serão fornecidas algumas instâncias válidas para serem utilizadas como entrada.

O algoritmo executará um número pré-definido de iterações, buscando encontrar o melhor agendamento possível. Ao final da execução, será apresentado o resultado do melhor agendamento encontrado, bem como será gerado um arquivo chamado "schedule.txt". Esse arquivo pode ser utilizado pelo arquivo "plot.py" para visualizar o agendamento por meio de um gráfico. Basta executar o arquivo "plot.py" para visualizar o agendamento gerado pelo algoritmo.

2. Métodos já empregados

A solução do problema de Job Shop Scheduling (JSP) tem sido alvo de diversas abordagens e métodos de resolução ao longo dos anos. Entre as técnicas mais comumente empregadas estão o aprendizado por máquina e as redes neurais.

Além disso, outros métodos heurísticos têm sido amplamente explorados para resolver o problema de escalonamento do JSP. Por exemplo, um dos artigos que serviu como base para este estudo propôs uma variante do método GRASP[1] (Greedy Randomized Adaptive Search Procedure) que incorpora um mecanismo de aprendizado. Essa adaptação permite ao GRASP aprender com os melhores agendamentos gerados anteriormente e utiliza essas informações para melhorar a qualidade das soluções encontradas.

Outro artigo implementou uma heurística multi-agente, combinando técnicas como Tabu Search e Algoritmo Genético[2]. Nessa abordagem, agentes independentes buscam soluções locais de alta qualidade e compartilham informações para explorar o espaço de busca de forma mais eficiente.

Esses métodos demonstraram resultados promissores na solução do JSP, contribuindo para o avanço do conhecimento nessa área. Neste trabalho, optou-se pela implementação de uma heurística genética para resolver o JSP.

3. Motivação

A escolha de implementar o algoritmo genético foi motivada pela alteraçao feita ao GRASP, especificamente em relação ao uso de um conjunto de elite para gerar novas soluções. Ao considerar essa ideia, surge a pergunta: por que não aproveitar uma heurística que tenha uma abordagem semelhante? O algoritmo genético se destaca nesse sentido.

Uma das vantagens do algoritmo genético é a sua capacidade de aproveitar as melhores características das soluções encontradas. Através da seleção dos melhores

indivíduos para compor a elite, é possível garantir que características valiosas sejam preservadas e transmitidas para as gerações futuras. Isso cria uma espécie de "aprendizado" no algoritmo, onde as melhores soluções são progressivamente aprimoradas.

Além disso, o algoritmo genético permite explorar o espaço de soluções de maneira mais abrangente. Ao combinar diferentes soluções e introduzir mutações, o algoritmo é capaz de buscar por soluções ainda melhores, levando em consideração as informações coletadas da elite. Essa abordagem ampla e exploratória contribui para a descoberta de soluções mais ótimas e eficientes para o problema de escalonamento do JSP.

4. Mapeamento do problema

No mapeamento do problema, foi crucial definir uma representação adequada para o agendamento, visando facilitar a operação de crossover no algoritmo genético. Foi estabelecido que uma tarefa k de um trabalho j só pode ser alocada no vetor após a tarefa (k-1) já ter sido alocada. Essa restrição garante a validade do agendamento.

Por exemplo, considerando um agendamento representado como { (1, 0), (2, 0), (2, 1), (1, 1)}, onde cada par representa um trabalho e sua respectiva tarefa k, a ordem de alocação segue a restrição mencionada. Essa representação do agendamento garante que todas as soluções geradas sejam válidas de acordo com as restrições do problema.

4.1. Calculando o custo de um agendamento

O custo de um agendamento é calculado utilizando as classes Machine e MachineController. A classe Machine representa cada máquina e processa as tarefas atribuídas a ela, enquanto a classe MachineController coordena o agendamento das tarefas, considerando as dependências entre elas. O tempo de espera entre as tarefas é determinado para garantir que as tarefas dependentes estejam prontas antes de serem processadas. Ao final do agendamento, o custo é calculado considerando o tempo total de processamento das máquinas.

5. Algoritmo Genético

5.1. Inicialização

No algoritmo genético implementado, o processo é iniciado com a geração de uma população inicial de agendamentos aleatórios. Em seguida, o custo de cada agendamento é calculado.

5.2. Crossover

No processo de crossover, dois pais são selecionados da população utilizando uma abordagem de roleta, onde a probabilidade de seleção é proporcional ao custo do agendamento. Em seguida, dois números, x e y, são sorteados, onde x é menor que y. A ordem das tarefas entre x e y é então realocada em um dos pais, seguindo a ordem em que elas aparecem no outro pai.

Após o crossover, é calculado o custo dos filhos resultantes. Se o custo for considerado bom o suficiente, o pior agendamento da população é removido e o filho é adicionado à população.

5.3. Mutação

A mutação é realizada através de um simples swap entre duas tarefas adjacentes nos filhos gerados pelo crossover. No entanto, é importante garantir que as tarefas selecionadas para a mutação sejam de jobs diferentes, de modo a preservar a validade do agendamento.

Essa operação de mutação introduz uma pequena perturbação nos agendamentos, permitindo explorar diferentes combinações de tarefas e potencialmente melhorar o desempenho da população.

6. Parâmetros

No algoritmo genético implementado, foram utilizados os seguintes parâmetros:

População: A população foi definida com o tamanho igual ao número de jobs multiplicado pelo número de máquinas. Cada indivíduo representa um agendamento possível para as tarefas.

Chance de Mutação: A chance de ocorrer uma mutação inicialmente começa com o valor de 1 e é multiplicada por 0.999 a cada iteração. Isso significa que a cada iteração, a chance de ocorrer uma mutação é ligeiramente reduzida.

Reinício da Mutação: Quando a chance de mutação atinge o valor de 0.1 após 2300 iterações, ela é reiniciada para o valor de 0.3. Esse reinício permite introduzir uma maior exploração do espaço de soluções mesmo após um longo período de execução do algoritmo.

Esses parâmetros foram escolhidos com base em experimentações e ajustes para obter um bom equilíbrio entre a exploração do espaço de solução e a convergência para melhores soluções.

7. Resultados

Para uma melhor comparação, utilizei as instâncias "abz" e "la" durante a avaliação dos resultados. Essas instâncias foram escolhidas com base nos artigos utilizados como referência, a fim de garantir uma comparação mais direta e consistente com os resultados relatados nessas publicações. Isso permitiu uma análise mais precisa da eficácia do algoritmo genético em relação ao desempenho e qualidade dos agendamentos encontrados nessas instâncias específicas do problema JSP.

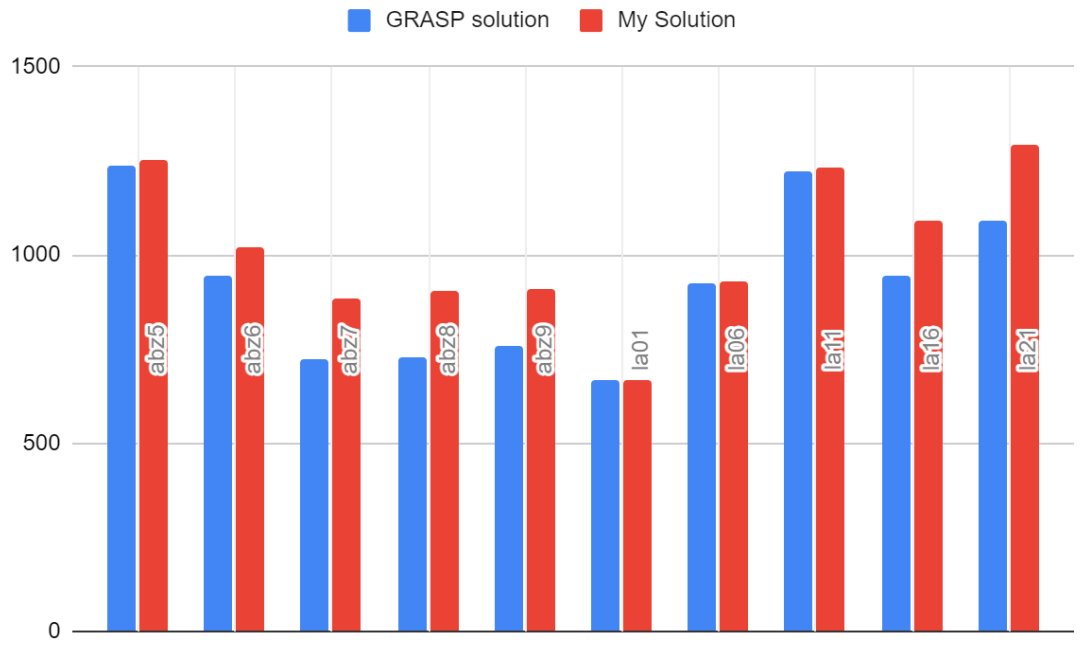
Os resultados obtidos pelo algoritmo genético implementado foram inferiores em comparação com o algoritmo GRASP. No entanto, é importante considerar também o tempo de execução necessário para obter esses resultados.

Por exemplo, no problema abz8, o algoritmo GRASP levou cerca de 5,5 horas para executar e fornecer um agendamento, enquanto o algoritmo genético implementado obteve resultados comparáveis em apenas 10 minutos.

Tabela com a comparação entre as instâncias:

Problem	Jobs	Machines	Iterations (10^6)	Time per 1000 iterations	My Iterations (10^6)	Time per 1000 iterations	GRASP solution	My Solution
abz5	10	10	20.1	0.3s	1	0.17s	1238	1254
abz6	10	10	20.1	3.1s	1	0.18s	947	1022
abz7	15	20	20.1	17.4s	1	0.57s	723	886
abz8	15	20	20.1	18.2s	1	0.55s	729	904
abz9	15	20	20.1	17.1	1	0.55s	758	909
la01	10	5	0.1	1.4s	1	0.1s	666	670
la06	15	5	0.1	2.4s	1	0.2s	926	930
la11	20	5	0.1	4.1s	1	0.2s	1222	1231
la16	10	10	0.1	3.1s	1	0.2s	946	1090
la21	15	10	0.1	6.5s	1	0.2s	1091	1291

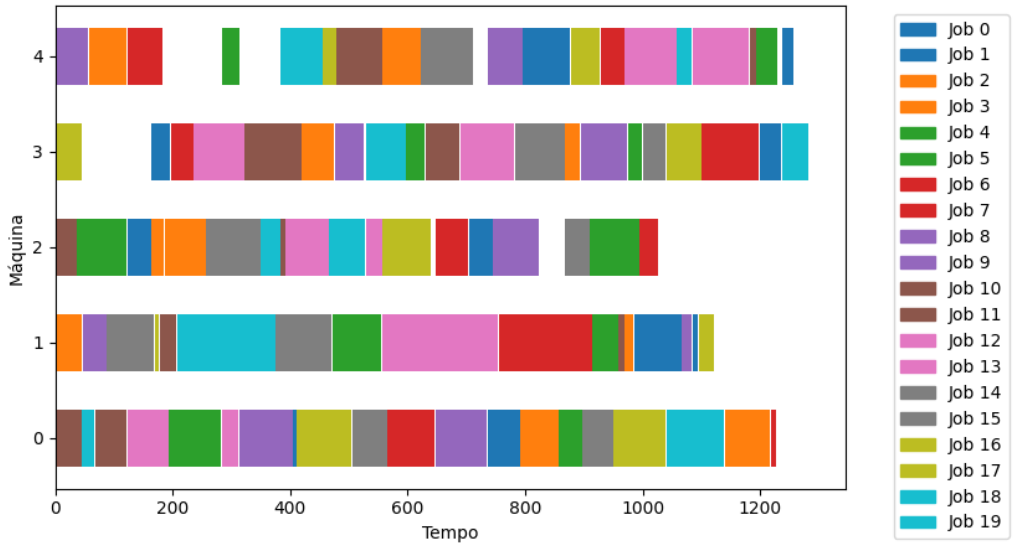
Visualizaçao em gráfico:



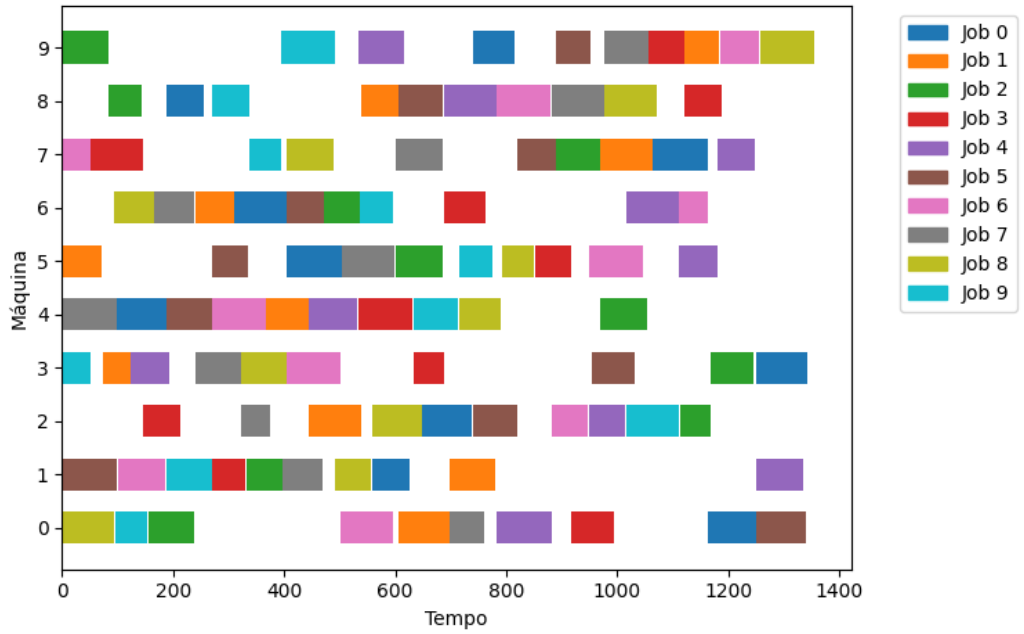
Em suma, o algoritmo genético apresenta uma vantagem distinta em relação ao tempo de execução em comparação com o GRASP, especialmente quando o número de máquinas é maior. Embora a diferença de desempenho em termos de custo seja observada, o algoritmo genético ainda é capaz de fornecer soluções razoáveis e competitivas, independentemente do aumento no número de jobs.

8. Visualização dos Agendamentos

O algoritmo genético implementado também possui uma funcionalidade de visualização do agendamento em forma de gráfico. Os gráficos abaixo são exemplos dessa representação visual, onde cada trabalho é identificado por uma cor distinta e o eixo Y representa as máquinas, enquanto o eixo X representa o tempo gasto. Essa visualização em forma de gráfico oferece uma maneira intuitiva e visualmente atraente de analisar o agendamento resultante, permitindo uma compreensão mais clara das interações entre as tarefas e as máquinas. Essa ferramenta auxilia na interpretação dos resultados e na identificação de possíveis áreas de melhoria no agendamento obtido.



Instancia la15



Instancia abz5

10. Conclusão

Em suma, a implementação do algoritmo genético para o problema de Escalonamento de Produção Job Shop Scheduling (JSP) apresentou resultados competitivos em termos de qualidade da solução e superou o algoritmo GRASP em relação ao tempo de execução. A heurística genética demonstrou eficácia na minimização do tempo total de produção e no cumprimento das restrições, gerando soluções consideradas boas dentro do tempo estipulado. Além disso, a visualização gráfica dos agendamentos proporcionou uma análise intuitiva e atraente das alocações de tarefas às máquinas, facilitando a interpretação dos resultados. Esses resultados evidenciam a aplicabilidade e o potencial do algoritmo genético para resolver problemas complexos de escalonamento de forma eficiente.

Referências

[1]: A GRASP FOR JOB SHOP SCHEDULINGS. BINATO, W.J. HERY, D.M. LOEWENSTERN, AND M.G.C. RESENDE: https://www.researchgate.net/publication/2475116_A_Grasp_for_Job_Shop_Scheduling

[2]: A multi-agents approach to solve job shop scheduling problems using meta-heuristics: https://www.sciencedirect.com/science/article/pii/S1474667016309004

[3]: Evolução do problema como tempo: https://en.wikipedia.org/wiki/Job-shop_scheduling

[4]: Instâncias: https://github.com/tamy0612/JSPLIB