UNIVERSIDADE FEDERAL DE MINAS GERAIS
ESCOLA DE ENGENHARIA
DEPARTAMENTO DE ENGENHARIA ELÉTRICA
ELE083 – COMPUTAÇÃO EVOLUCIONÁRIA

Alice Diniz Ferreira - 2014129473
Gabriel Saraiva Espeschit - 2015065541

TRABALHO PRÁTICO - JOB SHOP SCHEDULING PROBLEM

Belo Horizonte - MG
29 de novembro de 2019

1. Introdução

a. Job Shop Scheduling Problem

Em uma indústria a produção é realizada através da execução de uma sequência específica de processos. É comum que a mesma máquina seja compartilhada entre linhas de produção, variando-se o tempo de utilização dela de acordo com o produto final desejado.

Embora as máquinas possam ser compartilhadas, elas executam apenas um serviço por vez.

Dessa forma, a fim de minimizar o custo de tempo do processo completo de produção, é preciso definir a melhor ordem de execução de cada serviço. O Problema do Job Shop Scheduling consiste exatamente em definir esta ordem de execução que minimiza o makespan da produção utilizando-se técnicas de otimização combinatória.

Por tanto o problema pode ser definido como:

"Dadas X máquinas e Y serviços, determine a sequência de execução de serviços que demanda menos tempo para finalização de todos os produtos desejados."

Para o caso deste trabalho são dadas algumas restrições:

- uma mesma máquina não pode realizar duas tarefas simultaneamente;
- uma mesma tarefa não pode ser realizada por duas máquinas simultâneamente, ou seja, a tarefa só deve ser iniciada na máquina M_{m+1} após sua conclusão na máquina M_m;
- as tarefas devem ser processadas sequencialmente nas máquinas.

b. Objetivos

Esse trabalho tem como objetivo implementar um Algoritmo Genético capaz de resolver o JSSP.

2. Desenvolvimento

a. Inicialização dos Indivíduos

O cromossomo da solução é representado por um vetor de tamanho N, onde N é o número de jobs, contendo valores de 1 à N posicionados randomicamente. Esse conjunto de valores representa a ordem de execução das jobs.

Estes indivíduos são armazenados em uma matriz de M linhas, onde M é o número de indivíduos da população, setado como 40, formando assim uma matriz MxN.

Vide função "initializePopulation.m", anexada a este documento.

b. Inspiração

O algoritmo desenvolvido inspirou-se nas atividades desenvolvidas no Trabalho I (Algoritmo Genético para Problema das N-Rainhas) e no Trabalho II (Algoritmo Genético para Problema da Mochila) da disciplina de Computação Evolucionária e nos conceitos e técnicas aprendidos durante o semestre.

c. Cruzamento

O processo de cruzamento inicia-se com a escolha dos pais dos indivíduos. Para a escolha dos genitores se usou a seleção pelo método da roleta. Os indivíduos com os melhores fitness são dados preferência na seleção. 2 indivíduos são selecionados dessa forma para o cruzamento.

Já para o cruzamento de genes utilizou-se a função "CutAndCrossfill_Crossover", disponibilizada pelo professor no primeiro trabalho da disciplina.

Para cada iteração do algoritmo são gerados M (número total de indivíduos da população inicial) indivíduos.

d. Seleção

O método de seleção escolhido é um híbrido do método de Seleção Geracional e Seleção por Fitness: todos os indivíduos gerados na iteração são salvos em uma matriz auxiliar e após terminado o processo de cruzamento, o fitness dos novos indivíduos é comparado ao da população da geração anterior; caso o filho apresente um fitness inferior ao do pai, este substituirá o progenitor na próxima geração.

Vide função "JSSP.m".

e. Mutação

A mutação consiste na troca de posição de dois genes aleatórios do cromossomo.

Mutações são eventos probabilísticos, ou seja, sua ocorrência é determinada por uma probabilidade. A fim de simular este comportamento, para cada filho gerado, sorteia-se um número entre 0 e 1, caso o número sorteado seja inferior à probabilidade de mutação (definida como 0,2), haverá a troca na ordem de dois serviços da fila.

Vide função “JSSP.m”.

f. Critérios de Parada

Evitando que haja execuções desnecessárias do algoritmo, o mesmo é interrompido caso alguma das situações a seguir ocorra:

- O número máximo de gerações (400 gerações) seja alcançado;
- O menor tempo registrado se repita em 40 gerações seguidas.

Vide função “JSSP.m”.

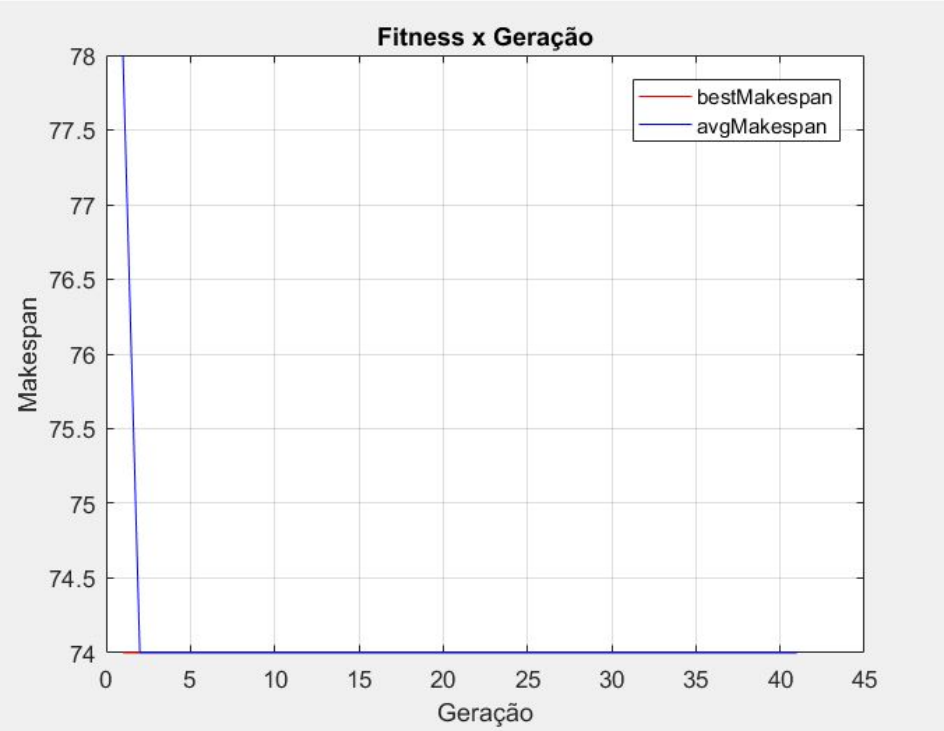
3. Resultados

A fim de validar o algoritmo desenvolvido, o mesmo foi executado múltiplas vezes para cada entrada fornecida pelo professor. O comportamento do algoritmo foi consistente para todas essas execuções. A seguir são apresentados os resultados para os últimos testes realizados.

a. Entrada 3 jobs

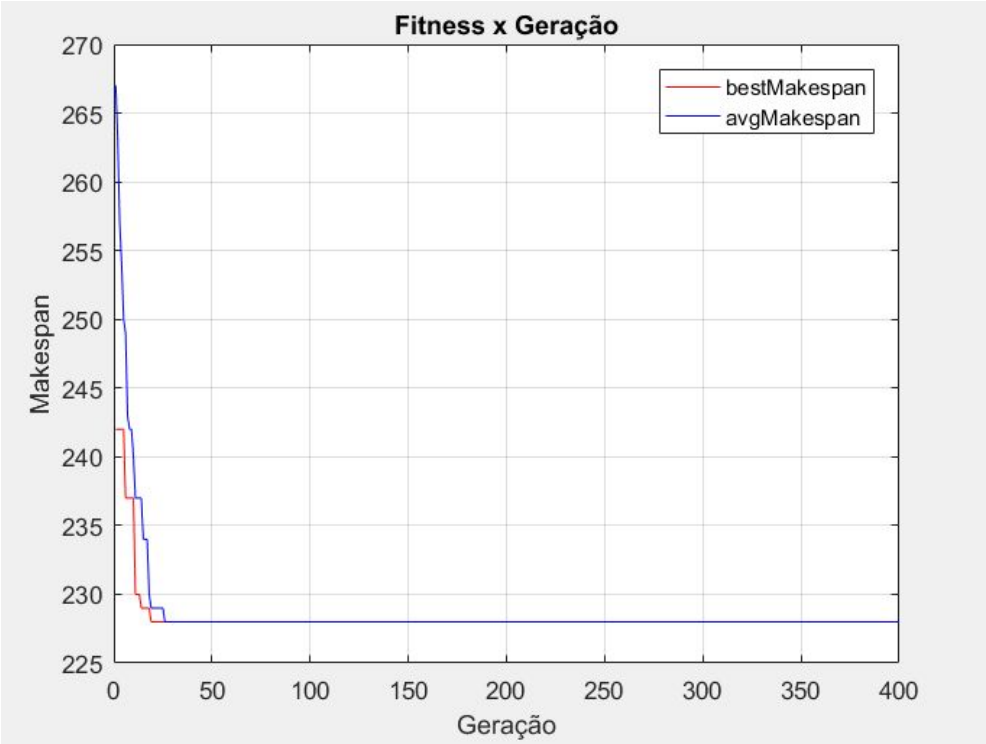
Para a entrada que contém três serviços obteve-se um tempo mínimo de produção igual à 74 minutos, dado para sequência [1, 2, 3].

Na figura abaixo está apresentada a relação do Fitness e da Geração do algoritmo, nota-se que quanto mais gerações são executadas mais os indivíduos da população convergem para o ponto ótimo encontrado. Observação realizada para todas as entradas sugeridas.



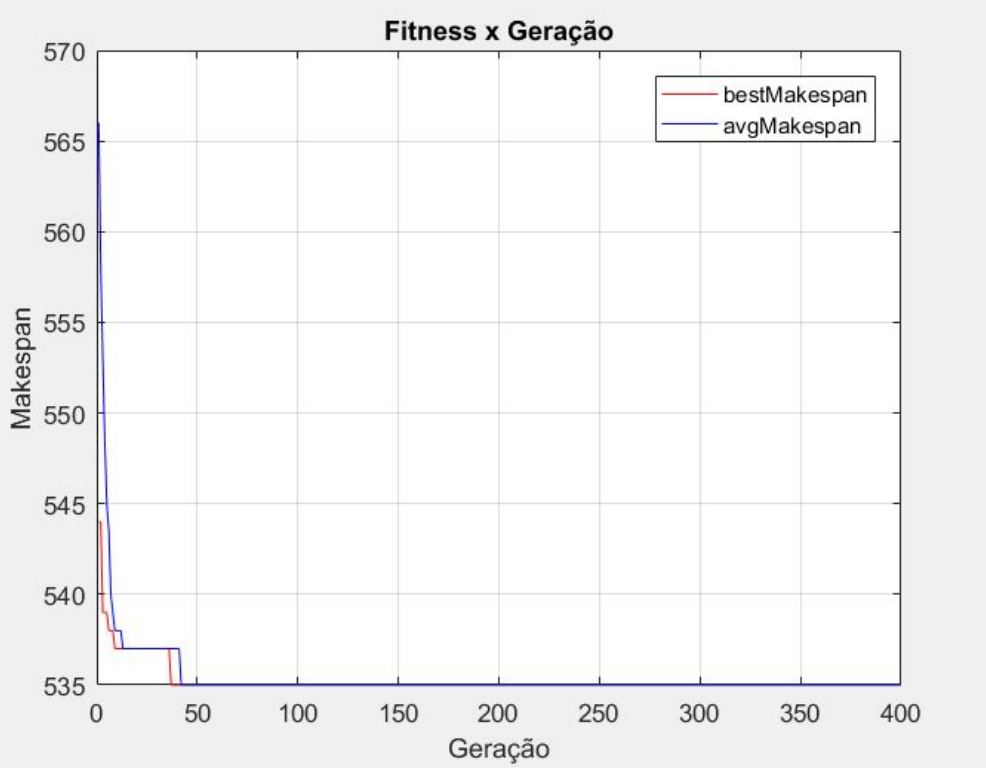
b. Entrada 10 jobs

Para a entrada que contém dez serviços a melhor sequência observada foi [10, 6, 4, 1, 7, 5, 3, 2, 8, 9], na qual o makespan é igual à 228 minutos.



c. Entrada 25 jobs

Por fim, para a entrada que contém vinte e cinco serviços a melhor sequência observada foi [7, 24, 3, 17, 2, 1, 6, 11, 20, 23, 12, 5, 18, 13, 4, 16, 8, 15, 22, 21, 10, 19, 25, 14, 9] e o menor makespan é igual à 573 minutos.



4. Conclusões

O trabalho objetivou a utilização de Algoritmos Genéticos para encontrar o menor tempo de execução de uma série de serviços em uma mesma linha de produção, o chamado Job Shop Scheduling Problem.

No decorrer do desenvolvimento do trabalho foram testados diferentes algoritmos e técnicas, como por exemplo Evolução Diferencial e seleção por roleta. Devido as dificuldades encontradas: definição dos limites do problema (para o algoritmo de Evolução Diferencial), não convergência da população (seleção por roleta), entre outros, através de análises de técnicas e resultados obtidos definiu-se adotar o desenvolvimento aqui documentado.

Além da dificuldade relacionada à definição do algoritmo, encontramos dificuldade em otimizar a função de cálculo do makespan. Porém nesse caso, não foi adotado o método mais indicado em fontes da internet: utilização de grafos para o cálculo do tempo de execução. Sendo este um possível ponto de melhoria para trabalhos futuros.

Os impasses encontrados nos permitiram ampliar nossos conhecimentos sobre algoritmos evolucionários, assim como sua aplicação em cenários mais próximos da engenharia.

Apesar das adversidades encontradas, obteve-se um algoritmo simples e com resultados condizentes com o esperado.

Sendo assim, os objetivos iniciais foram alcançados, demonstrando a validade da solução aplicada e do trabalho proposto.

5. Bibliografia

[1] Slides de sala de aula

[2] Algoritmo genético. *In*: **Wikipédia, a enciclopédia livre**. [s.l.: s.n.], 2019. Disponível em: <https://pt.wikipedia.org/w/index.php?title=Algoritmo_gen%C3%A9tico&oldid=55993399>. Acesso em: 29 dez. 2018.

[3] Job shop scheduling. *In*: **Wikipedia**. [s.l.: s.n.], 2019. Disponível em: <https://en.wikipedia.org/w/index.php?title=Job_shop_scheduling&oldid=926730775>. Acesso em: 29 dez. 2018.