UNIVERSIDADE FEDERAL DE MINAS GERAIS

ESCOLA DE ENGENHARIA

DEPARTAMENTO DE ENGENHARIA ELÉTRICA

COMPUTAÇÃO EVOLUCIONÁRIA

TRABALHO FINAL

JOB SHOP SCHEDULING PROBLEM

ALUNOS: BERNARDO SANTOS SUAREZ

FRANCIS CARLOS DOS SANTOS

VICTOR HUGO GONÇALVES DE OLIVEIRA

INTRODUÇÃO

JOB SHOP SCHEDULING

O Job Shop Scheduling é um método de maximização da utilização de alguns recursos produtivos (pessoas, máquinas, etc), onde procura-se uma forma ótima da distribuição desses recursos, de modo a obter o tempo total de produção mínimo (*makespan*).

Como esse problema é NP-Completo e não possui solução ótima em tempo factível, a ideia é gerar heurísticas de modo a se obter a melhor aproximação de um resultado ótimo. Esse modelo pode ser formulado de forma em que *n* tarefas e *m* máquinas concluindo um plano de produção em menor tempo possível e com a máximo tempo alocado.

FORMULAÇÃO DO PROBLEMA

TP FINAL

Para a solução apresentada nesse trabalho prático, alguns pontos devem ser considerados:

- Uma máquina só realiza uma tarefa por vez;
- Uma mesma tarefa não pode ser realizada por mais de uma máquina simultaneamente, além disso a tarefa M_{m+1} só pode ser iniciada após a conclusão de M_m;
- As tarefas devem ser processadas de forma sequencial.

A solução deve levar em conta 3 pedidos de produção, cada qual com seu tempo de processamento específico, e esses pedidos devem ser inseridos em uma ordem. O resultado final deve apresentar a ordem dos pedidos em que o tempo de produção seja o menor.

DESENVOLVIMENTO DA SOLUÇÃO

Para o problema do JSSP, o grupo optou por representar os indivíduos em um vetor de inteiros que vai de 1 até o número de planos de trabalho, onde cada posição do vetor representa um indivíduo, sendo cada indivíduo, um dos planos de trabalho.

Para tratamento dessa população, foi utilizado um Algoritmo Genético Geracional, que se baseia na substituição de indivíduos da população anterior por novos indivíduos de uma nova geração, que é algo que não representa gerações reais.

Nesse algoritmo genético, a estratégia de seleção de indivíduos utilizada foi a seleção por roleta. Essa técnica seleciona indivíduos de uma geração são escolhidos para fazer parte da próxima geração, através de um sorteio de roleta, onde cada indivíduo da população é representado na roleta proporcionalmente ao seu índice de aptidão. Sendo assim, quanto maior a aptidão do indivíduo, maior a chance de ele fazer parte da próxima geração. Quando classificados os indivíduos, a roleta é girada um determinado número de vezes, dependendo do tamanho da população, e são escolhidos os indivíduos que participarão da próxima geração.

Como método de cruzamento, foi utilizado o single point crossover que para este problema foi feito selecionando um determinado ponto. A partir disso, todos os *pais* eram copiados indo de 1 até este ponto e partir do ponto de cruzando só serão copiados os genes que não estão no vetor. Isso foi feito pois a

codificação foi feita considerando números inteiros e para a viabilidade da solução os trabalhos não poderiam ser repetidos

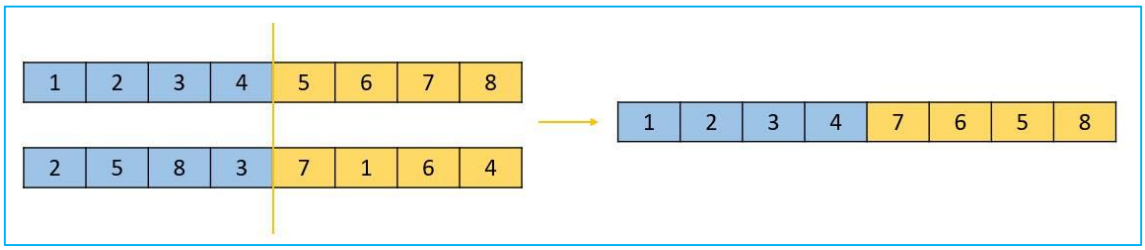

Figura 1 – Cruzamento utilizando Single Point Crossover

Para a mutação do algoritmo, foi feito um int Swap em que selecionamos duas posições no cromossomo de forma aleatória, e trocamos os valores. Esse tipo de permutação é bem comum em codificações baseadas em permutação.

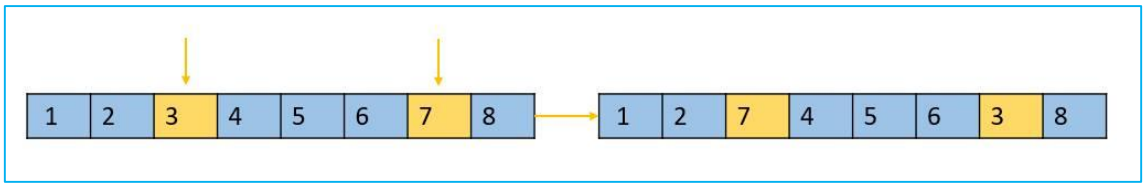

Figura 2 – Mutação baseada em swap

Os critérios de parada utilizados foram por número máximo de iterações, no nosso caso é 300, isso ocorreu, pois, a função fitness representava um gargalo para o sistema e para valores maiores que 300 o tempo de execução ficava alto para populações grandes. Além disso quando o melhor fitness era repetido o programa parava, pois era considerado uma condição para parada do sistema.

Nessa implementação foram utilizados alguns parâmetros importantes e que valem ser citados, por exemplo o tamanho da população 200 que foi utilizado para os testes. As probabilidades de cruzamento e mutação são 0,7 e 0,1 respectivamente.

ANÁLISE DOS RESULTADOS

Para análise dos resultados, foram utilizados os arquivos de entrada *entrada_3.txt*, *entrada_10.txt* e *entrada_25.txt*.

Gráfico Exemplo_3.txt

Sequência de entrada: [1 2 3]

Valor ótimo indicado pelo algoritmo: 74

Dados do exemplo:

Jobs	Máquina A	Máquina B	Máquina C
1	10	14	13
2	15	13	15
3	10	30	6

Tabela 1 – Roteiros de trabalho do exemplo_3.txt

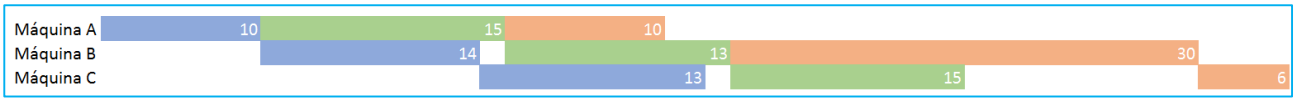

Figura 3 – Resultado das alocações para exemplo_3.txt

Melhor Valor indicado no gráfico do exemplo: 74

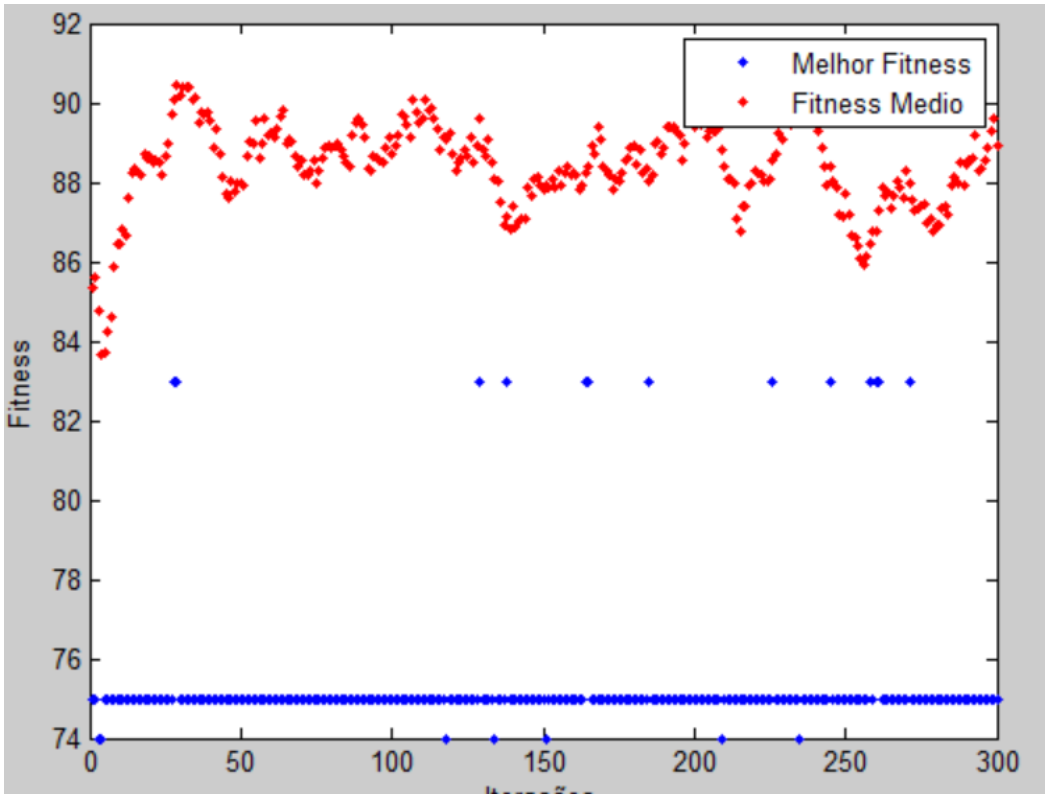

Figura 4 – Melhor fitness e fitness médio para cada iteração (300 iterações)

Gráfico Exemplo_10

Sequência de entrada: [9 1 10 8 6 4 7 2 3 5]

Makespan: 263

Dados do exemplo:

Jobs	Máquina A	Máquina B	Máquina C
1	11	22	29
2	26	30	13
3	19	16	17
4	9	29	9
5	20	25	28

6	8	6	30
7	30	28	26
8	30	17	26
9	23	26	12
10	6	10	19

Tabela 2 – Roteiros de trabalho do exemplo_3.txt

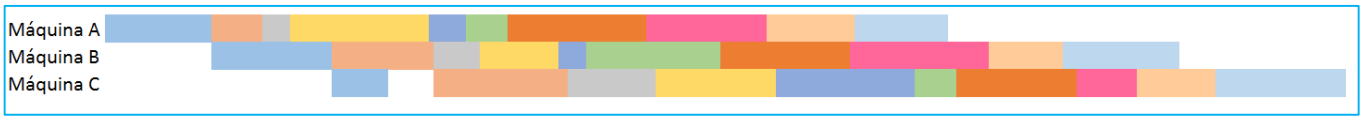

Figura 5 – Resultados das alocações para exemplo_10.txt

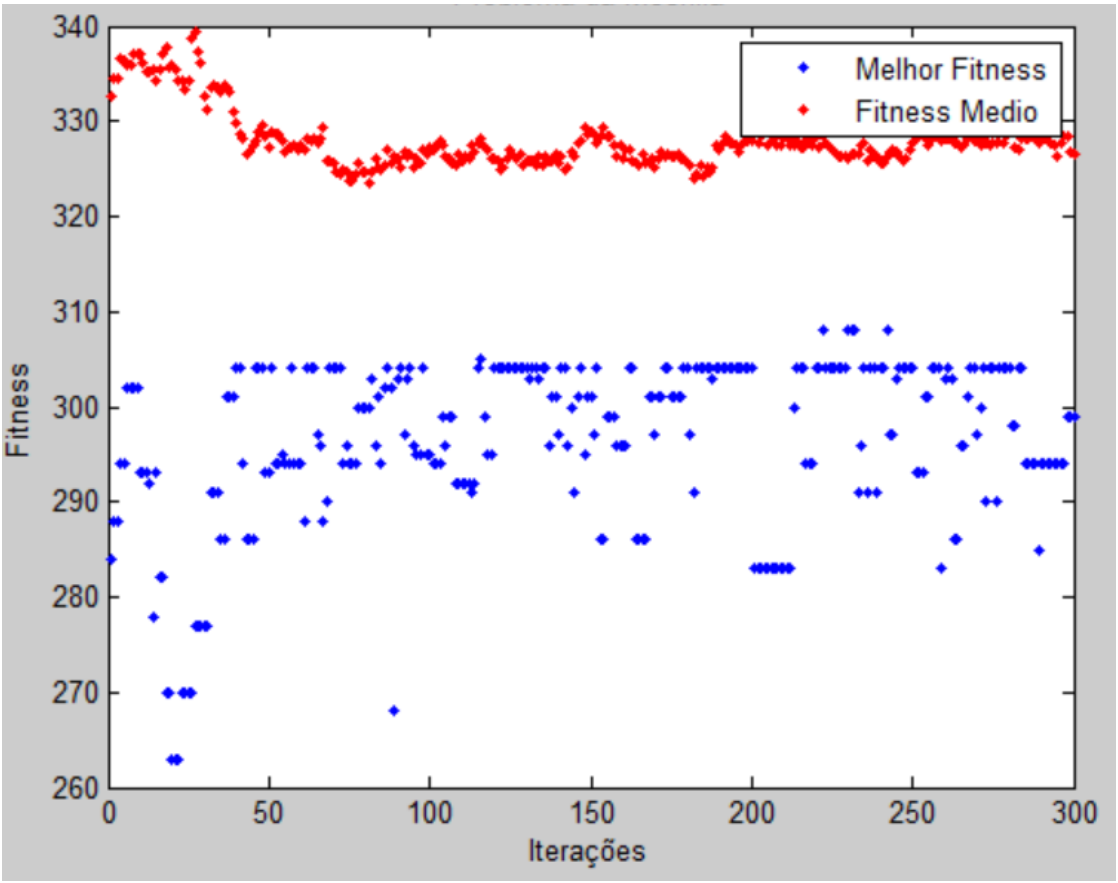

Figura 6 – Melhor fitness e fitness médio para cada iteração (300 iterações)

Gráfico Exemplo_25

Makespan = 582

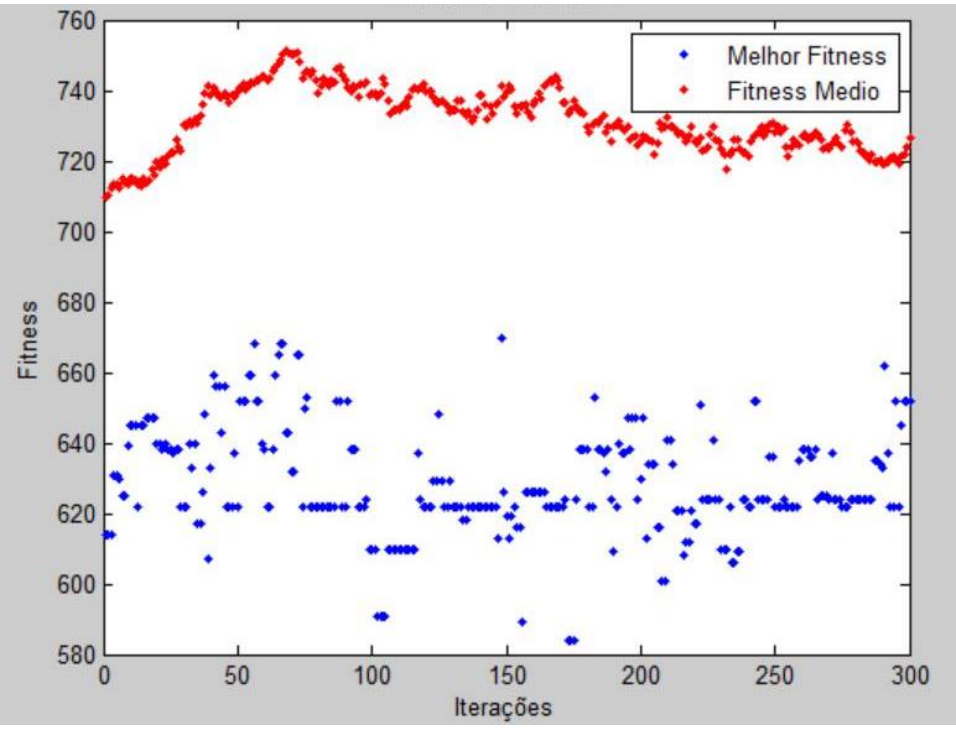

Figura 7 – Melhor fitness e fitness médio para cada iteração (300 iterações)

CONCLUSÃO

Conforme podemos perceber na resolução do problema JSSP, os melhores fitness indicaram o *makespan* que era o menor tempo para a conclusão de todos os trabalhos e apresentaram resultados bastante satisfatórios. Sendo assim os métodos aplicados foram adequados e comprovaram a eficiência esperada.

Acreditamos que uma melhoria na funções *fitness* nos permitiria atingir resultados melhores, pois como citado anteriormente, ela se tornou um gargalo no sistema pelo alto tempo de execução para grandes populações.

O grupo também crê a aplicação do método de seleção por roleta e de um algoritmo geracional foi uma escolha acertada, pois como a diferença entre as melhores soluções era pequena a chance de criar um super indivíduo era grande, o que é evitado com a aplicação da roleta. Para o caso do algoritmo geracional, o grupo considera uma boa decisão, pois não necessariamente indivíduos serão salvos para a próxima geração.

BIBLIOGRAFIA

http://www.w3ii.com/pt/genetic_algorithms/genetic_algorithms_mutation.html [w3ii.com]

Andries P. Engelbrecht, Computational Intelligence: An Introduction., John Wiley & Sons, Ltd, West, Sussex, England, 2008.

Joaquim de Oliveira Cruz, Adriano, Algoritmos Genéticos, Universidade Federal do Rio de Janeiro, Brasil, 05/2013