UFMG – Universidade Federal de Minas Gerais

TRABALHO DE COMPUTAÇÃO EVOLUCIONÁRIA

JOB SHOP SCHEDULING PROBLEM APLICADO A LAMINAÇÃO

JOÃO PEDRO SAMARINO

PEDRO HENRIQUE LEÃO BRAGA

Belo Horizonte, 2016

João Pedro Samarino, Pedro Henrique Leão Braga

Computação Evolucionária Job Shop Scheduling Problem Aplicado a Laminação

Trabalho referente a utilização de algoritmos evolucionários para resolver o problema do tipo Job Scheduling aplicado a laminação na disciplina de Computação evolucionária, no curso de Engenharia de Sistemas, na Universidade Federal de Minas Gerais.

Professor: Cristiano Leite

Belo Horizonte, 2016

1. INTRODUÇÃO

Job Shop Schdulin Problem (JSSP) é um problema clássico de otimização combinatória e possui diversas aplicações nas indústrias e empresas.

Ele consiste em obter uma sequencias de tarefas a serem executadas de forma a maximizar a utilização de recursos disponíveis.

O JSSP é um problema NP-Completo, ou seja não possui algoritmo que o resolva em tempo polinomial, dessa forma a abordagem de algoritmos evolucionários pode ser uma boa solução para encontrar uma boa resposta em tempo hábil.

O problema a ser resolvido neste trabalho é do tipo JSSP aplicado ao processo de laminação onde existem 3 maquinas (forno de reaquecimento, laminador, bobinadeira), e cada um dos produtos precisam passar pelas 3 maquinas e em cada uma delas existe um tempo de processamento, o nosso objetivo é dado uma lista de produtos reordena-los de tal forma que o tempo gasto para a confecção de todos seja o menor possível, respeitando algumas restrições que são:

- Um equipamento não pode processar dois pedidos simultaneamente.
- Um mesmo pedido não pode ser processado simultaneamente por mais de um dos 3 equipamentos.
- Para o mesmo pedido o material deve ser processado necessariamente nesta ordem: forno, laminador, bobinadeira.

2. DESENVOLVIMENTO

2.1 Forma de Representação dos indivíduos

Na figura abaixo podemos visualizar 5 produtos a serem transformados, seus respectivos tempos de cada máquina e seus respectivos números:

Tabela 1 Tabela de Produtos e Tempo de Maquina

Produto	Tempo de Máquina		
Nº Produto	Tempo maquina 1	Tempo maquina 2	Tempo maquina 3
1	21	26	19
2	26	25	21
3	22	27	21
4	26	11	7
5	7	25	15

Para este exemplo, cada indivíduo (I_n) gerado na população será um vetor de 5 posições onde cada posição do vetor indica a ordem de processamento do produto exemplo:

$$I_1 = [\,3\ \ 1\ \ 2\ \ 5\ \ 4]$$

Neste caso o primeiro produto a ser processado é o produto 3, o segundo produto a ser processado é o produto 1, assim por diante até chegar no produto 4 que foi o último a ser processado.

2.2 Inspiração do algoritmo Evolucionário

Este algoritmo foi baseado em um algoritmo genético, para problemas combinatórios, sua representação é por meio de combinações de números inteiros reais positivos, conforme detalhado anteriormente. A ideia por trás da evolução da população média é fundamentada na ideia de que um bom indivíduo pode gerar outros bons indivíduos.

2.3 Operador de Seleção

O operador de seleção utilizado neste problema foi um operador pseudoaleatório, o seu funcionamento se baseia em gerar dois vetores com os indivíduos aleatório sem repetição e cruza-se os indivíduos do primeiro vetor com o segundo vetor, isso promove a diversidade dos filhos.

Tabela 2 Explicação do Método de seleção dos Pais

Indivíduos	Vetor aleatório 1	Vetor aleatório 2	Cruzamento
1	2	1	2 x 1
2	4	3	4 x 3
3	1	2	1 x 2
4	3	4	3 x 4

2.4 Operador de Cruzamento

O operador escolhido foi o Order crossover devido sua fácil implementação e ele gera um bom desempenho, abaixo pode-se ver uma figura explicando o processo.

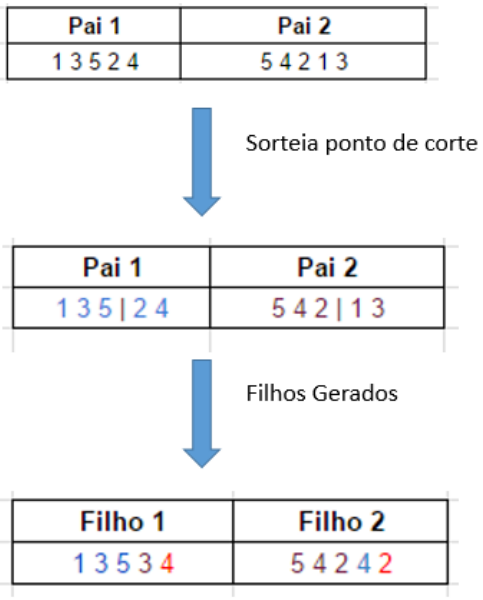

Figura 1 Desenho explicativo modelo Order Crossover

Conforme podemos ver na figura acima, dados os pais sorteia um ponto de corte em seus genes, após isso faz-se um crossover entre o pai 1 (azul) e o pai 2(roxo) da seguinte maneira, pegando o exemplo do filho 1 podemos perceber que os 3 primeiros genes são do pai 1, a partir daí completa-se os genes do filho 1 na ordem que eles aparecem no Pai 2, no entanto existe um gene no Pai 2 que não aparece para ser completado no Filho 1, o gene 4 dessa forma completa-se o filho 1 com o gene 4(em vermelho).

2.5 Operador de Mutação

Bit-flip alterado, se realiza 8,5% de mutação repetida 12 vezes ou seja, escolhe-se com uma taxa de 8,5%, indivíduos da população de forma aleatória e sem repetição, e se realiza um flip de 2 cromossomos de posições aleatórias, esse processo se repete esse procedimento 12 vezes, esse número foi estimado através de experimentos.

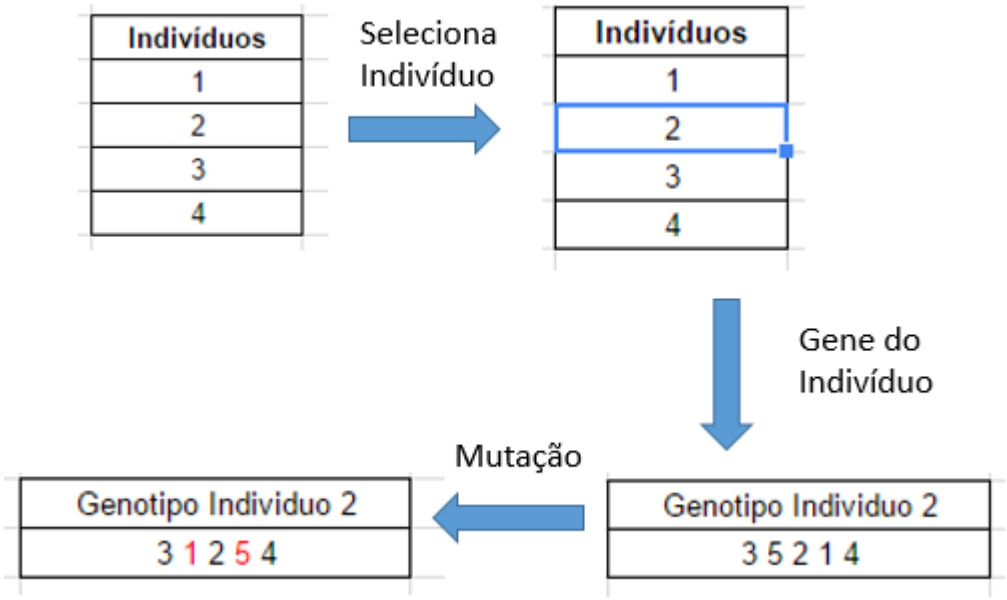

Figura 2 Explicação Mutação Bit-Flip Alterado

Esse procedimento é repetido 12 vezes

2.6 Critério de parada

O critério de parada utilizado foram 1000 gerações que permanecem em um mesmo resultado ótimo, obtido através de resultados experimentais.

2.7 Parâmetros do algoritmo

Os parâmetros do algoritmo foram modelados de acordo com vários testes realizados anteriormente, para defini-los da melhor maneira possível.

População: 50 indivíduos

Geração: 50 indivíduos

Cruzamento: 50 indivíduos

Taxa de mutação: 8,5%

2.8 Outras informações

A seleção dos sobreviventes é feita da seguinte forma, os pais são comparados com filhos de índices equivalentes, isso não quer dizer que um determinado pai será comparado com seu determinado filho, e sim um pai será comparado com um filho aleatório, de modo que o melhor individuo dessa comparação permanecerá, é importante ressaltar que o melhor indivíduo é sempre guardado para que o mesmo não sofra mutação dessa forma o melhor individuo nunca sofre mutação, ou seja o algoritmo é elitista

O algoritmo apresentado até o momento foi a solução final(JSSP) desenvolvido para a resolução do problema, outros 2 algoritmos foram implementados anteriormente utilizando diferentes formas, para solucionar o problema. Todas as representações dos indivíduos para os 3 algoritmos sãs as mesmas utilizadas na solução final(JSSP), descrito na seção 2.1.

JSSP1:

Inspiração do algoritmo evolucionário: Algoritmo genético com baixo cruzamento por geração

Operador de seleção: Torneio de 5 indivíduos aleatórios

Operador de Cruzamento: Order cross over

Operador de Mutação: bit-flip

Critério de parada: 3000 iterações

População: 40 indivíduos

Geração: 2 indivíduos

Cruzamento: É realizado apenas 1 cruzamento por geração

Taxa de mutação: 40%

Observações:

São separados 20 indivíduos que são inertes, ou seja, ele só são substituídos quando os mesmos se tornam o pior o ou o penúltimo pior individuo, o que gera vantagem para problemas com curvas de níveis acentuadas ele tem maior possibilidade de alcança-las, devido seu baixo cruzamento e alta mutação,

JSSP2:

Inspiração do algoritmo evolucionário: Algoritmo genético

Operador de seleção: torneio de 5 indivíduos

Operador de Cruzamento: Order cross over

Operador de Mutação: bit-flip alterado com parâmetros de mutação dinâmica em relação a geração atual

Critério de parada: 3000 iterações

População: 20 indivíduos

Geração: 10 indivíduos

Cruzamento: 5 cruzamentos

Taxa de mutação: dinâmica

Observações:

A mutação ocorre da mesma forma que na solução final(JSSP) com a alteração de seus parâmetros, isso se dá da seguinte forma: a seleção da taxa de mutação é dada pela função $Taxa\ Mutação = -0{,}00011\ x\ Z + 0.4$, onde Z é o número de iterações, e essa mutação, assim como na solução final(JSSP) se repetia por 12 vezes, neste caso ela se repete de acordo com uma outra função que é dada por $Número\ Repetição\ Mutação = 0{,}0037\ x\ Z + 1$, onde Z é o número de iterações, dessa forma temos o seguinte exemplo:

$$Taxa\ Mutação = -0{,}00011\ x\ Z + 0.4$$

$$Número\ Repetição\ Mutação = 0{,}0037\ x\ Z + 1$$

Nº Iterações	Taxa Mutação	Nº Repetição
0	0.4	1
1	0.39989	1.0037
500	0.345	2.85
1000	0.29	4.7
2000	0.18	8.4
3000	0.07	12.1

Figura 3 Exemplificação das funções aplicadas ao Bit-Flip

Como podemos perceber a taxa de mutação vai diminuindo ao longo do número de iterações e o número de aplicação de mutações vai aumentando, dessa forma ele é vantajoso porque percorre diferentes taxas de mutação e o algoritmo é elitista para não perder a melhor solução.

Um comportamento curioso deste algoritmo é a forma dos indivíduos médios que tem comportamento parabólico, uma vez que a multiplicação das 2 funções de repetição de aplicação da mutação e Taxa de mutação gera uma função de segundo grau, dessa forma podemos verificar esse comportamento nas figuras 6, 9 e 12 deste trabalho.

3. RESULTADOS

Os resultados serão apresentados para os 3 algoritmos descritos acima, de tal forma que a solução final(JSSP) é o algoritmo descrito nos itens 2.1 a 2.7, e o JSSP1 e o JSSP2 foram descritos no item 2.8, dessa forma tem-se:

Entrada 3 :

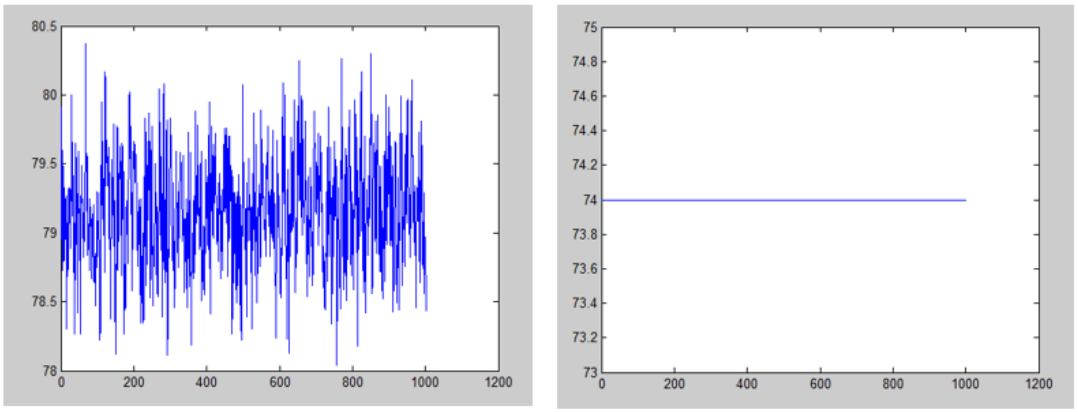

Gráfico Individuo médio

Gráfico Melhor indivíduo

Figura 4 Resultado do Gráfico Makespan x Gerações para 3 entradas solução Final(JSSP)

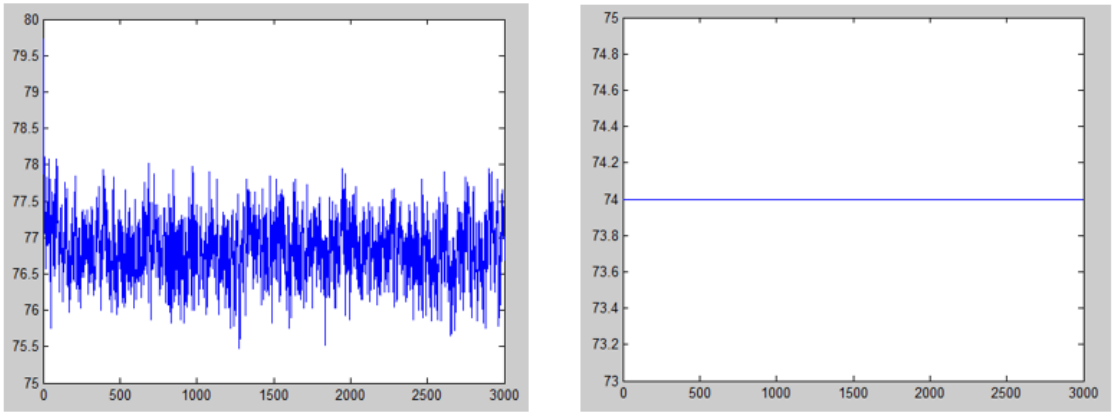

Gráfico indivíduo Médio

Gráfico Melhor indivíduo

Figura 5 Resultado do Gráfico Makespan x Gerações para 3 entradas JSSP1

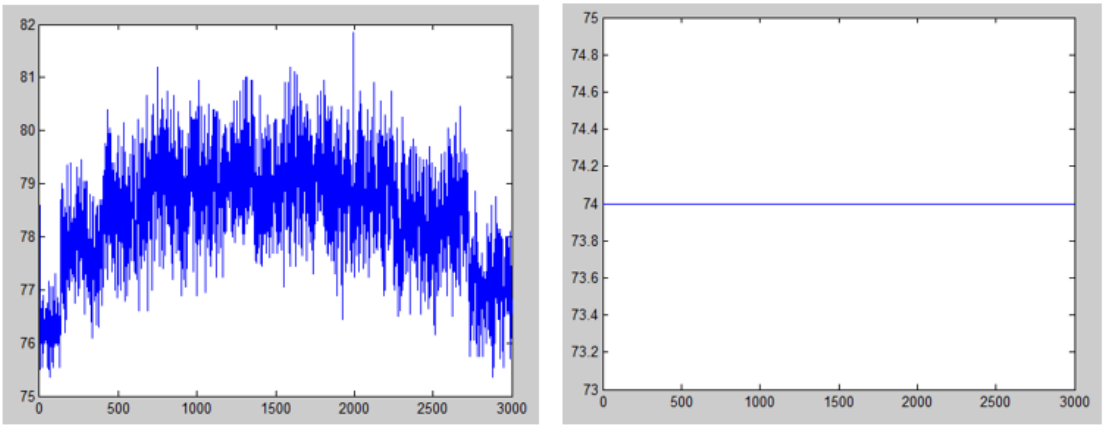

Gráfico Individuo médio

Gráfico Melhor indivíduo

Figura 6 Resultado do Gráfico Makespan x Gerações para 3 entradas JSSP2

Entrada 10:

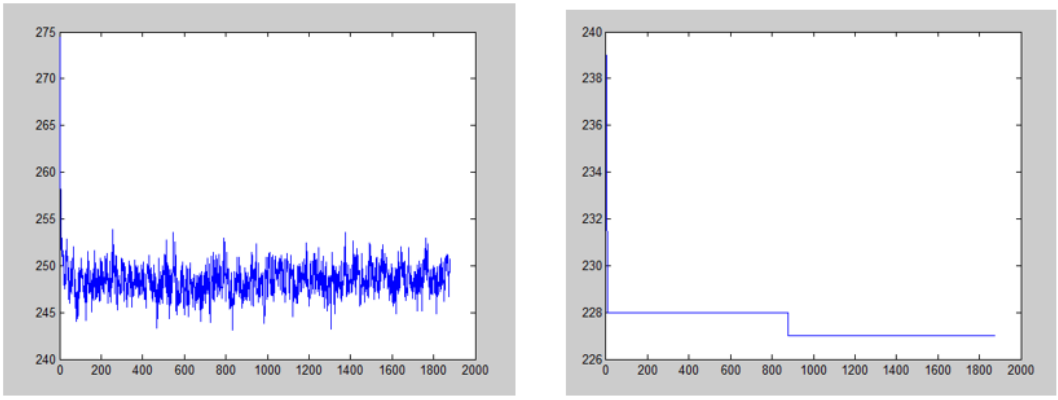

Figura 7 Resultado do Gráfico Makespan x Gerações para 10 entradas Solução Final(JSSP)

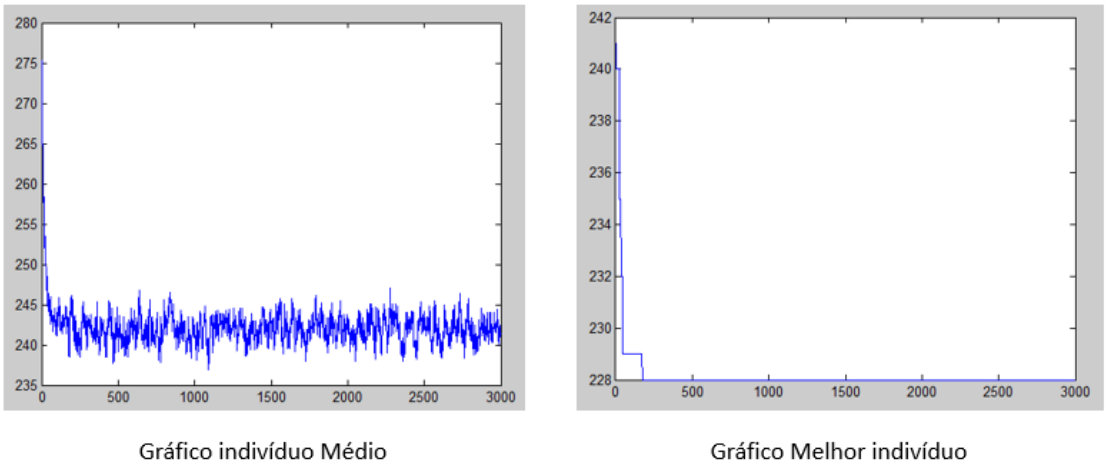

Figura 8 Resultado do Gráfico Makespan x Gerações para 10 entradas JSSP1

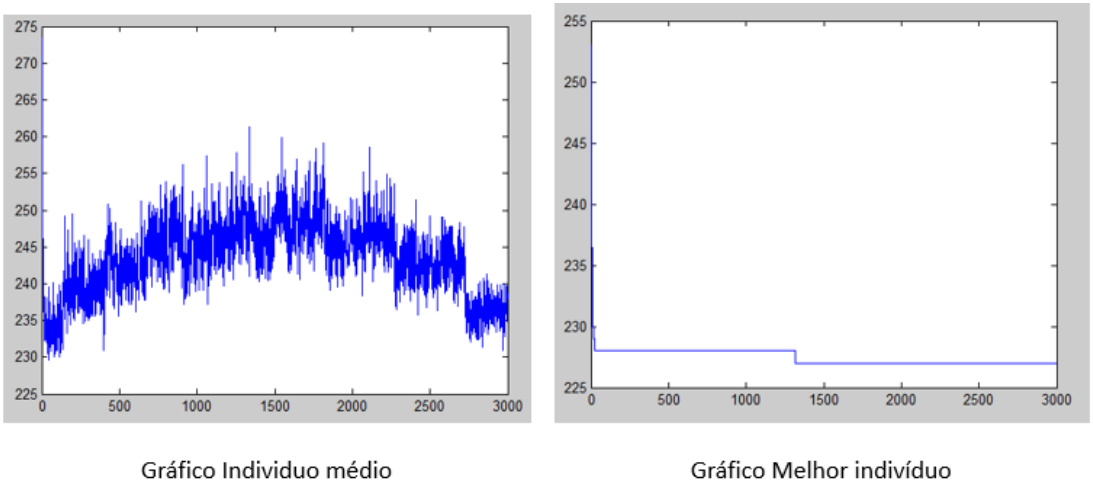

Figura 9 Resultado do Gráfico Makespan x Gerações para 10 entradas JSSP2

Entrada 25:

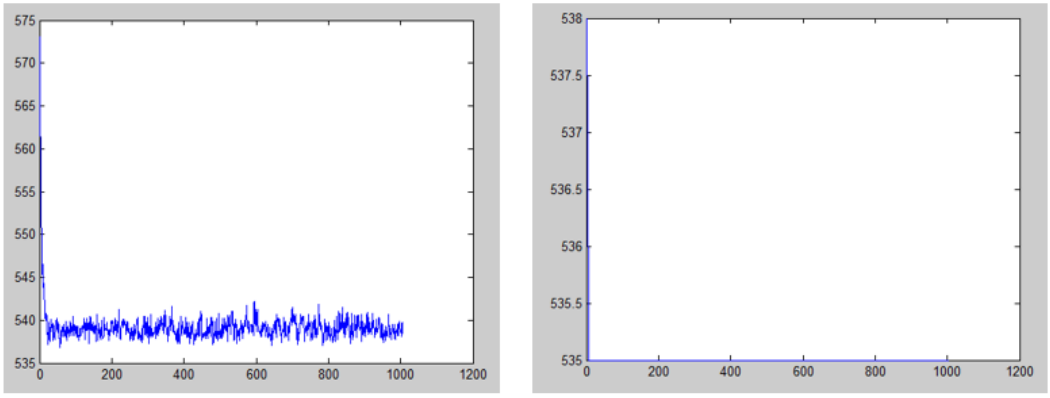

Gráfico Individuo médio

Gráfico Melhor indivíduo

Figura 10 Resultado do Gráfico Makespan x Gerações para 25 entradas Solução Fina(JSSP)l

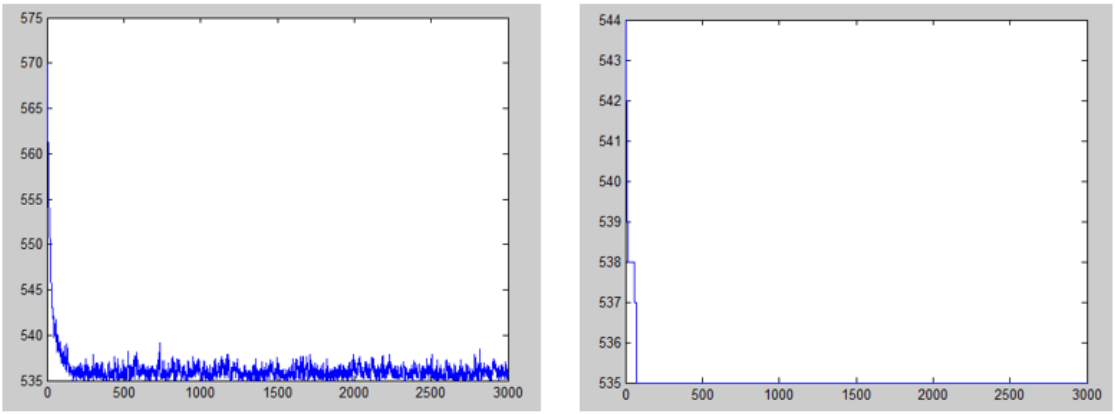

Gráfico indivíduo Médio

Gráfico Melhor indivíduo

Figura 11 Resultado do Gráfico Makespan x Gerações para 25 entradas JSSP1

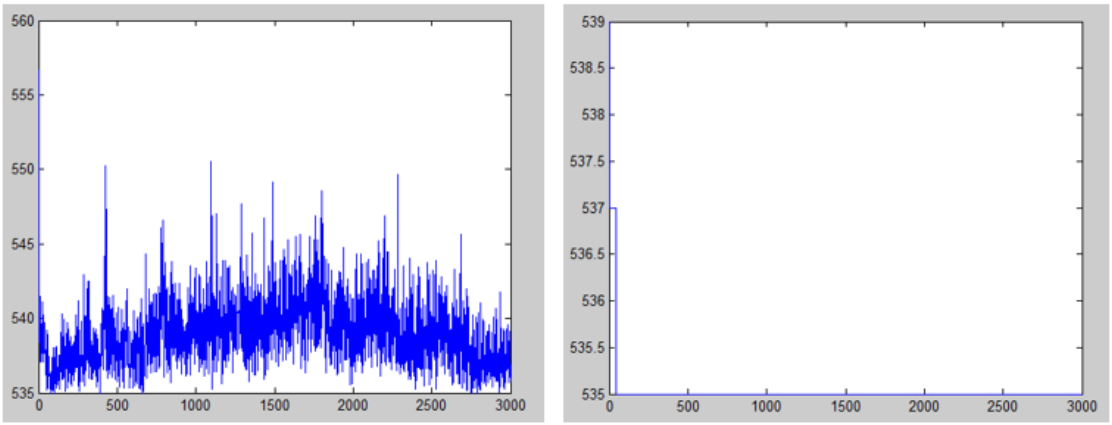

Gráfico Individuo médio

Gráfico Melhor indivíduo

Figura 12 Resultado do Gráfico Makespan x Gerações para 25 entradas JSSP2

Para os 3 casos é possível perceber que a solução final(JSSP) convergiu mais rapidamente e de maneira correta em relação aos outros 2 algoritmos, percebemos o JSSP2 convergir prematuramente para o valor 228 para 10 entradas, e devido as condições de mutação presentes no mesmo o comportamento do seu indivíduo médio tem comportamento parabólico, explicado anteriormente. Apesar do JSSP1 convergir mais rápido houveram casos em que o mesmo não conseguiu convergir para 10 entradas, diferentemente da solução final(JSSP) que garantiu convergência em 100% dos testes.

4. CONCLUSÃO

Através desse trabalho foi possível verificar a eficiência de resolver problemas com algoritmos genéticos. Conseguimos resolver um problema NP-Completo de maneira rápida e chegando ao resultado ótimo, provando a importância dos estudos nessa área.

Foi possível perceber também que dependo da abordagem escolhida para tratar os problemas o algoritmo pode ou não convergir para o ponto ótimo, e o tempo para atingi-lo leva pode variar bastante, comprovando que conhecer bem o problema e as técnicas de algoritmos genéticos garantem melhores maneiras de resolver o problema desejado.

5. BIBLIOGRAFIA

As bibliografias utilizadas foram os slides disponibilizados pelo professor Cristiano no Moodle.