Influência da taxa de mutação em Algoritmos Genéticos no Problema da Mochila

Bruno Corrêa Feil, Jessica Rocha

21 de outubro de 2018

1 Introdução

O problema da mochila consiste em uma mochila com um peso limite e um conjunto de itens que possuem pesos e valores associados. O objetivo é colocar itens na mochila de forma que o valor seja maximizado e que o peso limite da mochila não seja ultrapassado. Podemos mapear o problema da mochila para algoritmos genéticos da seguinte forma:

Um indivíduo vira uma classe que contém:

- Pesos de todos os itens e os seus respectivos valores;
- Peso limite da mochila;
- Nota da avaliação do indivíduo (fitness);
- Espaço ocupado pelos itens presentes na solução do indivíduo e
- Cromossomo (lista de caracteres contendo o valor "0", que significa que o iésimo item não pertence à solução do indivíduo, ou o valor "1", que significa que o iésimo item pertence à solução do indivíduo).

Uma população vira uma classe que contém:

- O tamanho da população (Quantos indivíduos a população tem);
- A lista de indivíduos da população;
- A geração da população e
- O melhor indivíduo já encontrado até então. Isto é, o indivíduo com melhor fitness encontrado.

Na abordagem dos Algoritmos Genéticos há o conceito de mutação, abordaremos sobre a sua implementação na sessão 3. A taxa de mutação, define a chance de algum bit do cromossomo ser "mutado", ou seja, ser modificado (caso o bit que sofrerá a mutação tenha o valor "0", irá conter o valor "1"após a

mutação. Acontece de forma análoga com bits que tenham o valor "1"e sofram mutação).

Como o problema da mochila é um problema relativamente simples, pensamos em analisar a influência que a taxa de mutação exerce sobre os resultados, ao invés de tentar mudar alguma coisa em relação ao problema em si. Com isso, nosso objetivo com o trabalho é o de analisar a influência da taxa de mutação em algoritmos genéticos, mais especificamente no problema da mochila. A pergunta proposta é: "Valores muito altos na taxa de mutação geram resultados piores?".

Nossa hipótese é que se a taxa de mutação for muito alta, os resultado serão prejudicados.

2 Trabalho Relacionado

O artigo que escolhemos para pegar os dados é o artigo intitulado "An Improved Genetic Algorithm for 0-1 Knapsack Problems" dos autores Wei Shen, Beibei Xu e Jiang-ping Huang disponível através do link:

https://ieeexplore.ieee.org/document/6047101

Neste artigo os autores mostram três abordagens em relação a algoritmos genéticos.

A primeira abordagem nos mostra um algoritmo genético com estratégia elitista. Com ele, a população é dividida em castas. A casta com os melhores indivíduos sempre passará para a próxima geração. A casta com indivíduos piores que os melhores mas melhores que os piores (ou seja, a casta mediana) irá fazer crossover com os indivíduos da melhor casta. Os indivíduos da pior casta ou são eliminados, ou então fazer crossover com os indivíduos da melhor casta, varia conforme a implementação. No artigo de Shen et al. [2011], não há menção sobre o que é feito com os piores indivíduos.

A segunda abordagem nos mostra um algoritmo genético com duas populações. Nele, as duas populações são desenvolvidas separadamente mas os melhores indivíduos são compartilhados. Em certa altura do algoritmo, alguns indivíduos das duas populações fazem crossover. Este método de duas populações ajuda a aumentar diversidade de indivíduos e, uma consequência direta disso é, fugir de soluções ótimas locais.

A terceira abordagem é chamada apenas de "algoritmo genético com duas populações melhorado". Essa abordagem tem algumas melhorias em relação à segunda abordagem apresentada. As populações tem probabilidades de crossover e mutação diferentes. Com probabilidades de crossover diferentes de 100%, os filhos nem sempre existirão, porque os pais nem sempre "terão filhos", às vezes o que seguirá para a próxima geração são os próprios pais. Quando há probabilidades de crossover diferentes entre duas populações, uma dela terá mais filhos passando para a próxima geração (maior diversidade genética) enquanto a outra população terá mais pais passando para a próxima geração (menor diversidade genética). Em uma analogia com o mundo real: é como se uma população se

reproduzisse mais de forma sexuada e outra população se reproduzisse mais de forma assexuada. Isso pode ser benéfico para o algoritmo genético como um todo porque, enquanto a população "assexuada" mantém mais os melhores indivíduos, a população "sexuada" está mais livre para encontrar novos indivíduos.

Na terceira abordagem há também o chamado de "Critério Guloso" em que os itens da mochila são ordenados através da divisão do seu valor pelo seu peso:

$$Iten_i = \frac{Valor_i}{Peso_i}$$

A partir deste ordenamento, o algoritmo genético irá priorizar a seleção de itens com melhor relação entre valor/peso.

2.1 Resultados Mostrados no Artigo

Tabela 1: Resultados experimentais de algoritmo genético com duas populações, algoritmo genético com estratégia elitista e algoritmo genético com duas populações melhorado

algorithm	optimum / volume	worst value / volume	average value	Average of spending time s
elitist strategy genetic algorithm	2990/1000	2898/1000	2940	0.0516
dual population genetic algorithm	3103/1000	3019/1000	3072	1.19256
Improved dual population genetic algorithm	3093/1000	2967/1000	3054.367	0.756

3 Materiais e Métodos

3.1 Sobre os dados

Os dados foram encontrados no artigo de Shen et al. [2011] e não foram pré-processados. Os dados consistem em 50 itens, seus respectivos valores e pesos e o peso limite da mochila. São eles:

$$\begin{aligned} Valores_i = 220, 208, 198, 192, 180, 180, 165, 162, 160, 158, \\ 155, 130, 125, 122, 120, 118, 115, 110, 105, 101, \\ 100, 100, 98, 96, 95, 90, 88, 82, 80, 77, \\ 75, 73, 72, 70, 69, 66, 65, 63, 60, 58, \\ 56, 50, 30, 20, 15, 10, 8, 5, 3, 1, \end{aligned}$$

$$\begin{aligned} Pesos_i = 80, 82, 85, 70, 72, 70, 66, 50, 55, 25, \\ 50, 55, 40, 48, 50, 32, 22, 60, 30, 32, \\ 40, 38, 35, 32, 25, 28, 30, 22, 50, 30, \\ 45, 30, 60, 50, 20, 65, 20, 25, 30, 10, \\ 20, 25, 15, 10, 10, 10, 4, 4, 2, 1, \end{aligned}$$

$$Limite = 1000$$

3.2 Operadores Utilizados

3.2.1 Linguagem da Implementação

Decidimos implementar o algoritmo genético em R porque a linguagem possui muitos recursos que auxiliam a implementação, além de ter uma sintaxe clara e ser uma linguagem com alto nível de abstração.

3.2.2 Cálculo do Fitness

Para o calculo do fitness, nosso algoritmo percorre o cromossomo e soma todos os valores (e pesos) dos elementos presentes na solução. Se o peso for maior que o limite de peso da mochila, a avaliação terá o valor igual a 1.

3.2.3 Seleção de Pais

Para a seleção de pais utilizamos o método da roleta.

3.2.4 Mutação

A nossa implementação da mutação consiste em percorrer o cromossomo de cada indivíduo e, para cada bit do cromossomo, gerar um valor aleatório entre 0 e 1. Se esse valor aleatório for maior do que a taxa de mutação, o bit é modificado de “0” para “1” ou de “1” para “0”.

3.2.5 Crossover

Utilizamos 100% como sendo nossa taxa de crossover. Ou seja, todos os indivíduos que participarem do crossover, irão gerar filhos (nunca haverá pais indo para a próxima geração). Na nossa implementação, o crossover se dá com um corte em um único ponto do cromossomo. Não há restrição sobre o ponto do corte (assim, os filhos poderão ser iguais aos pais).

3.2.6 Sobre as Escolhas dos Operadores

Como nosso foco é na taxa de mutação e suas implicações, decidimos deixar o nosso algoritmo genético o mais padrão possível. Ou seja, utilizando operadores com implementações simples.

3.3 Configuração dos Experimentos

Como queríamos comparar os nossos resultados com os obtidos pelo artigo de Shen et al. [2011] decidimos usar uma configuração muito parecida com a configuração utilizada pelos autores do artigo para que pudéssemos comparar melhor. Nossa configuração dos experimentos foi:

- Número de gerações: 200 (experimentos também foram feitos com 400 gerações, mas os resultados estão na apresentação de slides com link na sessão 3.4)
- Tamanho do cromossomo: 50;
- Tamanho da população: 100;
- Probabilidade de crossover: 100%;
- Limite de peso da mochila: 1000 e
- Utilizamos 9 valores de taxa de mutação. São eles:
    1. 0%
    2. 1%
    3. 2.5%
    4. 5%
    5. 10%
    6. 25%
    7. 50%
    8. 70%
    9. 90%

3.4 Links

Link para nossa apresentação em slides:

https://docs.google.com/presentation/d/1BotDlefWrH0N76eiz4Cw1xx9o-9MH1TR3BubWeA86Nc/edit?usp=sharing

Link para nossa implementação(contém a apresentação em slides também):

https://github.com/brunofeil/ComputacaoEvolutiva-AG

4 Resultados

Para análise dos resultados fizemos uma plotagem dos resultados de duas formas:

1. Valores médios dos fitness de cada geração e
2. Valores dos melhores indivíduos de cada geração.

Como ficaram muitos gráficos (e muitos deles com resultados redundantes) deixamos todos os gráficos dos resultados na nossa apresentação de slides (disponível através do link na sessão 3.4) e mostraremos aqui apenas os gráficos mais relevantes. Os gráficos mais relevantes são os de taxa de mutação = 0%, 1%, 2.5%, 5%, 10%, 25% e 50%.

4.1 Valores médios e melhores indivíduos de Cada Geração

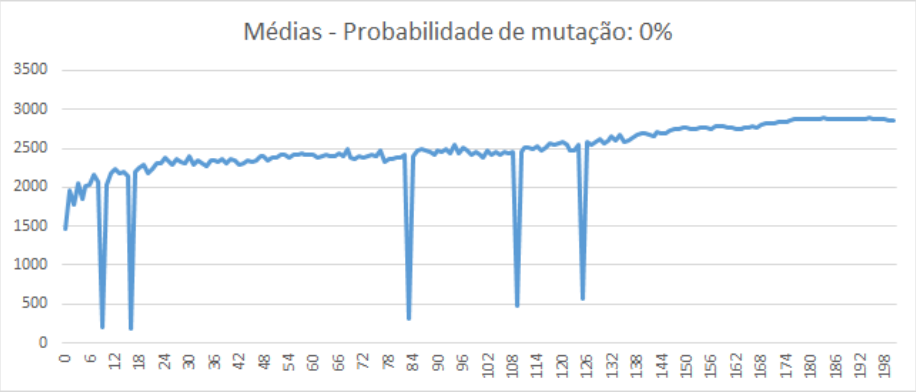

Figura 1: Gráfico do valor médio dos fitness dos indivíduos em relação às gerações - 0% taxa de mutação

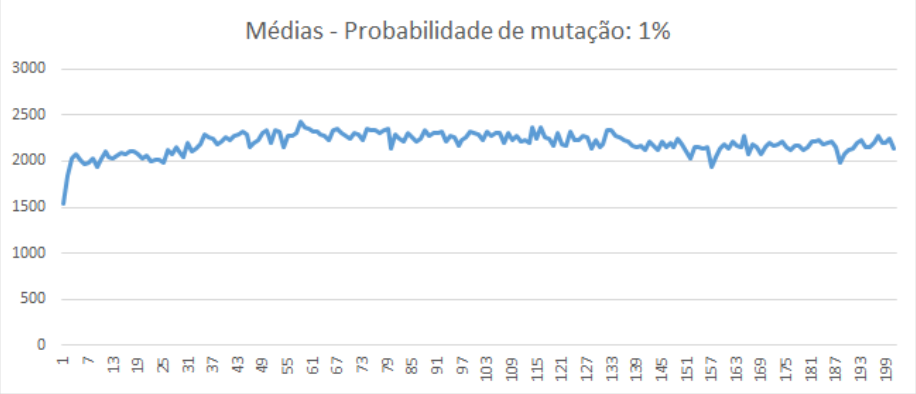

Figura 2: Gráfico do valor médio dos fitness dos indivíduos em relação às gerações - 1% taxa de mutação

4.2 Análise dos Resultados

Analisando os gráficos dos valores médios dos fitness, vemos que com taxa e mutação igual a 0% a média dos fitness tendo a aumentar com o aumento das gerações até a geração 180 mais ou menos. Após esta geração, a média dos fitness permanece a mesma até a última geração. Acreditamos que isso se deu porque, como a mutação é uma maneira de fugir de ótimos locais e a taxa de mutação era de 0%, as soluções acabaram caindo em um ótimo local sem

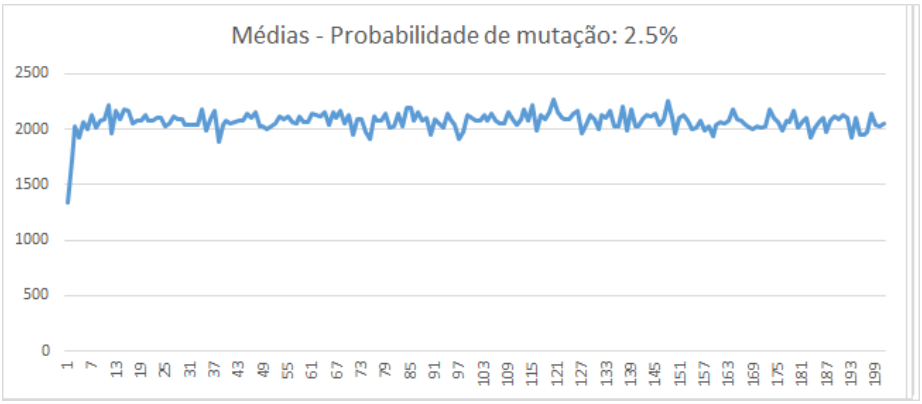

Figura 3: Gráfico do valor médio dos fitness dos indivíduos em relação às gerações - 2.5% taxa de mutação

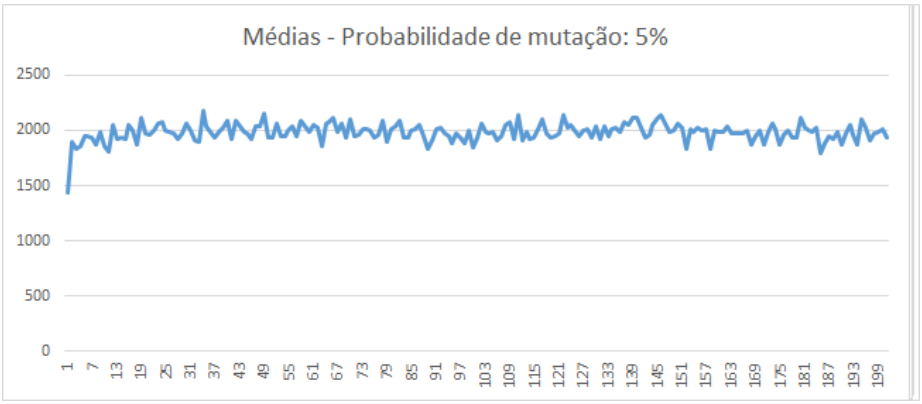

Figura 4: Gráfico do valor médio dos fitness dos indivíduos em relação às gerações - 5% taxa de mutação

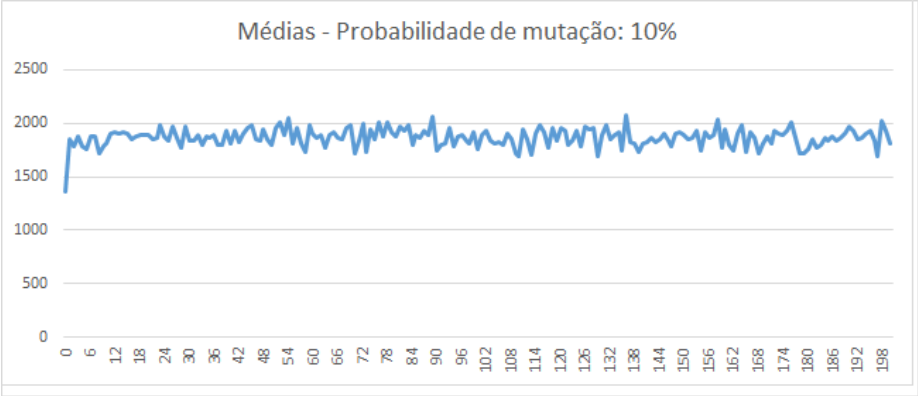

Figura 5: Gráfico do valor médio dos fitness dos indivíduos em relação às gerações - 10% taxa de mutação

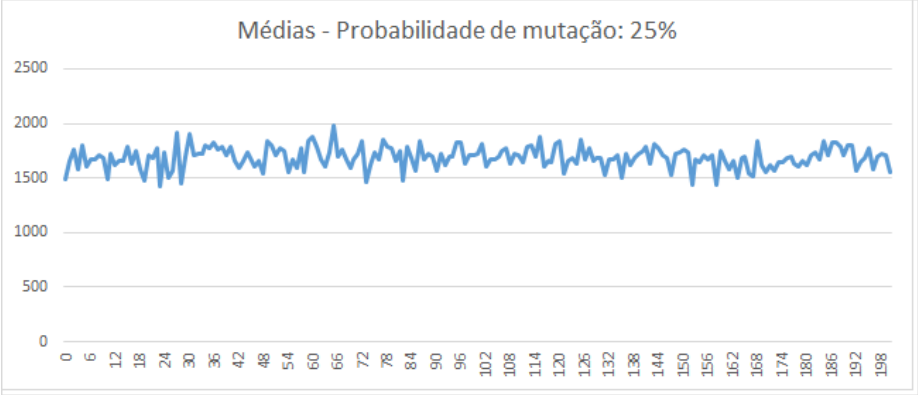

Figura 6: Gráfico do valor médio dos fitness dos indivíduos em relação às gerações - 25% taxa de mutação

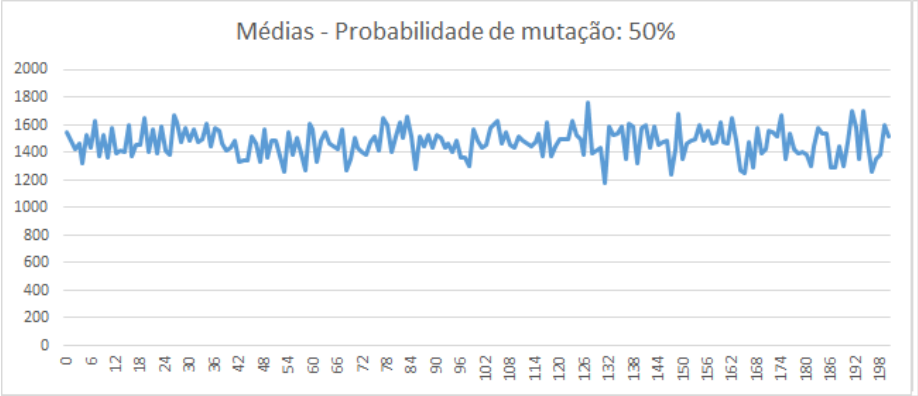

Figura 7: Gráfico do valor médio dos fitness dos indivíduos em relação às gerações - 50% taxa de mutação

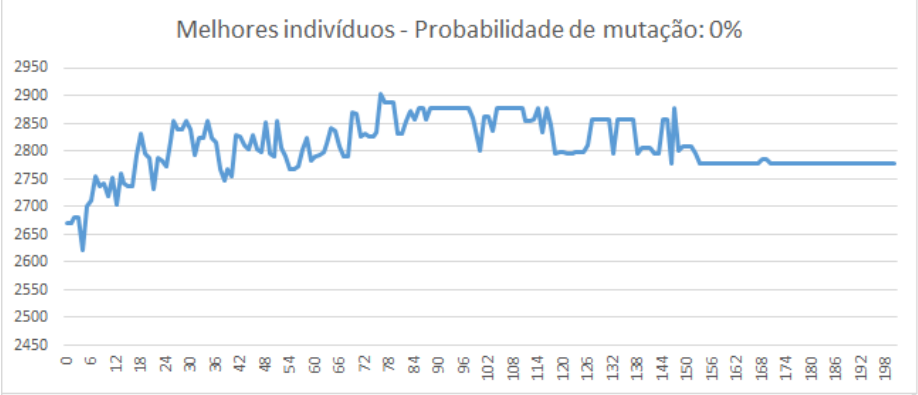

Figura 8: Gráfico do melhor indivíduo em cada geração - 0% taxa de mutação

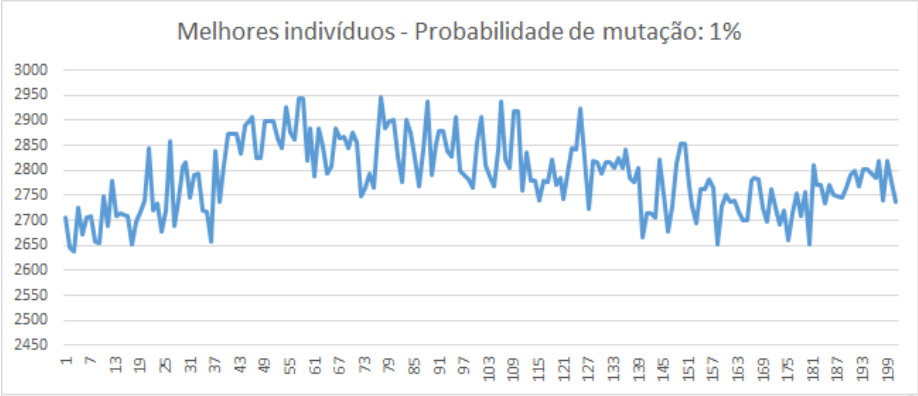

Figura 9: Gráfico do melhor indivíduo em cada geração - 1% taxa de mutação

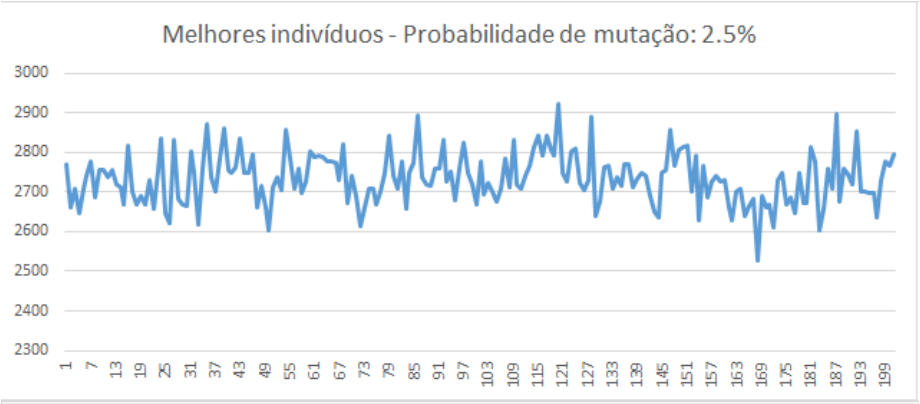

Figura 10: Gráfico do melhor indivíduo em cada geração - 2.5% taxa de mutação

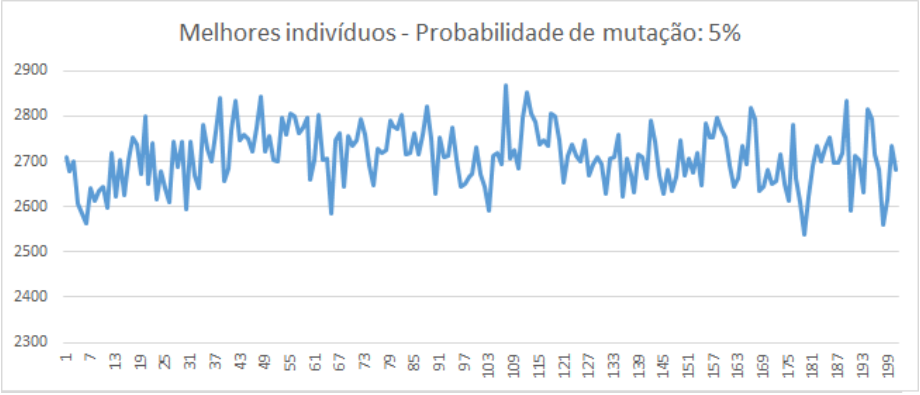

Figura 11: Gráfico do melhor indivíduo em cada geração - 5% taxa de mutação

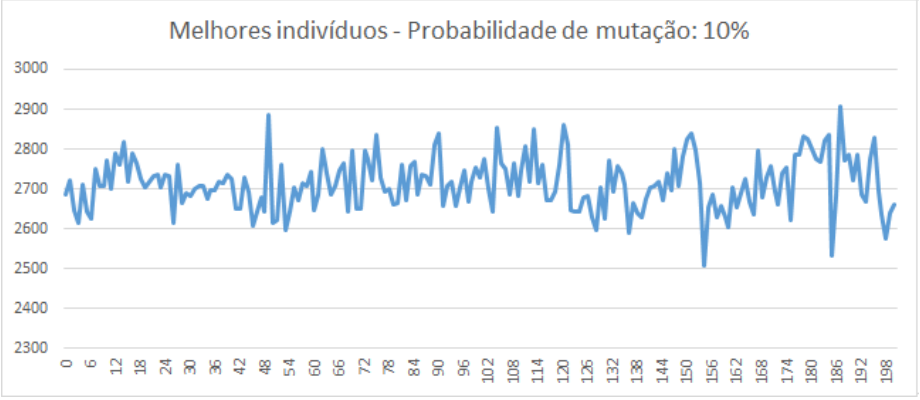

Figura 12: Gráfico do melhor indivíduo em cada geração - 10% taxa de mutação

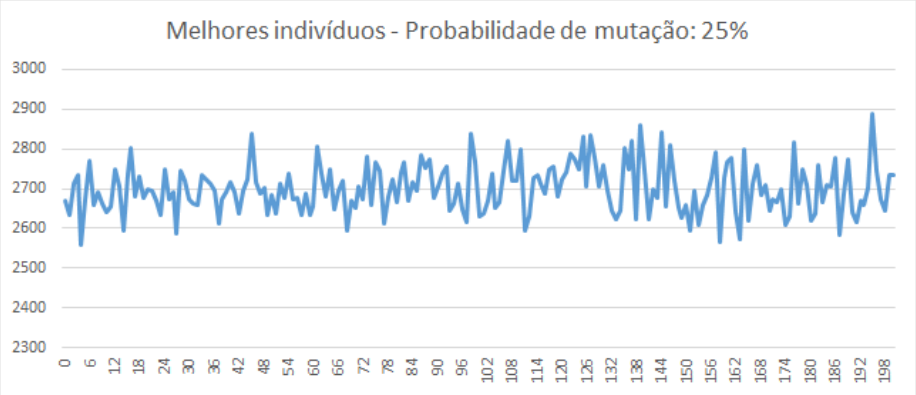

Figura 13: Gráfico do melhor indivíduo em cada geração - 25% taxa de mutação

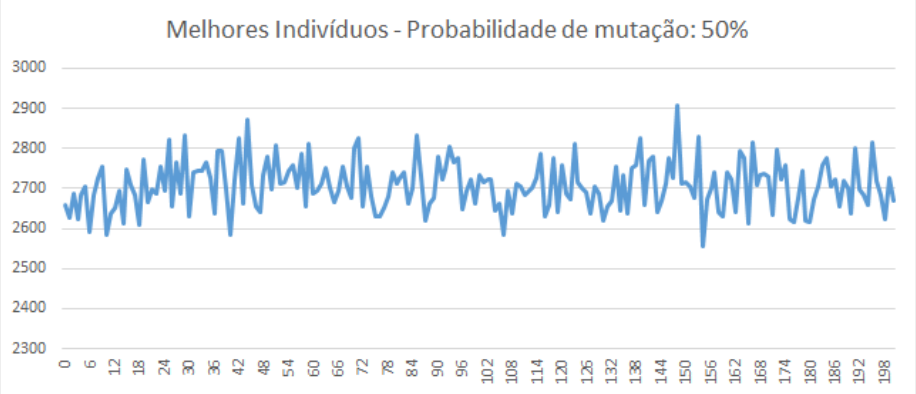

Figura 14: Gráfico do melhor indivíduo em cada geração - 50% taxa de mutação

conseguir sair. Com a taxa de mutação sendo igual a 1%, vemos que não ocorre o mesmo que o exemplo anterior. Não há uma estabilização das médias, mas os seus valores ficam entre 2000 e 2500, valores menores do que o visto no gráfico anterior. O mesmo ocorre com gráficos da taxa de mutação igual a 2.5%, 5%, 10%, 25% e 50%, mas quanto maior é a taxa de mutação, menor é o valor das médias dos fitness.

Analisando os gráficos dos melhores indivíduos por geração, vemos que com taxa de mutação igual a 0% ocorre o mesmo que ocorreu com o gráfico dos valores médios. Após a geração 156 mais ou menos, o melhor indivíduo em cada geração não muda. Vemos claramente que é um ótimo local porque na geração 78 mais ou menos, achamos um indivíduo que é o melhor encontrado pela solução. Ou seja, o melhor indivíduo da geração 156 até a 200 é pior que o encontrado na 78. Aumentando a taxa de mutação, este comportamento não volta a acontecer. O que acontece é que conforme se aumenta a taxa de mutação, a frequência com que indivíduos bons (com soluções acima de 2900) aparecem diminui.

4.3 Comparação

Colocando nossas melhores soluções em uma tabela (Tabela 2) e comparando as nossas melhores soluções com as soluções de Shen et al. [2011] (presente na tabela 1) temos que mesmo com nossa melhor solução (2946) chegamos perto da pior solução encontrada no artigo (2990), mas bem longe da melhor (3103). Isso mostra que as otimizações feitas pelos autores do artigo é realmente muito eficaz.

5 Conclusões

Nossa hipótese (vista na sessão 1) foi confirmada. Se o valor da taxa de mutação for muito alto, acabará piorando os resultados. Na verdade, a taxa de mutação não precisa ser muito alta para que os resultados acabem piorando. Notamos também que, como nosso cromossomo é muito grande (50 itens) a taxa de mutação tem mais influência ainda. Mesmo com uma taxa relativamente pequena, a influência nos resultados é muito grande. A função da mutação é dar um grau de aleatoriedade no algoritmo (esse grau tende a ser pequeno) e, com isso, fugir um pouco dos ótimos locais. Se aumentarmos muito a taxa de mutação, nossas soluções ficarão muito aleatórias. Com isso, fugiremos sim dos ótimos locais, mas fugiremos também dos ótimos globais, porque o algoritmo ficará muito aleatório.

Tabela 2: Resultados experimentais da nossa implementação do algoritmo genético

Taxa de mutação	0%	1%	2.50%	5%	10%	25%	50%
Melhores soluções	2905	2946	2922	2867	2908	2888	2909

Os algoritmos genéticos tem alguns mecanismos para fugir de soluções ótimas locais(como o crossover entre os pais, a seleção dos pais que leva em conta todos os indivíduos e não apenas os melhores), mas a taxa de mutação é um mecanismo muito importante. Como vimos, quando a taxa de mutação é igual a 0%, as soluções tendem a chegar em ótimos locais e dificilmente conseguirão sair deles.

Também percebemos que, na seleção dos pais usando o método da roleta, quanto maior for o tamanho da população, menos impacto o fitness do melhor indivíduo terá na hora de escolher os pais.

6 Bibliografia

Referências

Wei Shen, Beibei Xu, and Jiang-ping Huang. An improved genetic algorithm for 0-1 knapsack problems. *Second International Conference on Networking and Distributed Computing*, pages 32–35, oct 2011.