Algoritmo Genético Aplicado ao Problema do Caixeiro Viajante

Matheus Henrique Batista Santos[1], **Antonio Carlos Martins Manhes Filho**[2]

[1]Departamento de Ciência da Computação
Universidade Federal de São João del-Rei (UFSJ)

matheusrickbatista@aluno.ufsj.edu.br[1], juniormanhaesfilho@aluno.ufsj.edu.br[2]

Abstract. *The traveling salesman problem (TSP) is a well-known combinatorial optimization problem. Its simplest version consists of determining the shortest route to travel through a series of cities, visiting them only once, returning to the city of origin. It has not yet been possible to prove that the problem has an optimal solution in polynomial time. Thus, there is a tendency to use approximate methods to solve it. This work presents the application of one of these methods, known as genetic algorithm (GA). GA is a metaheuristic of the evolutionary algorithm class that uses techniques inspired by evolutionary biology. Such techniques find good approximations and, sometimes, optimal results for TSP.*

Resumo. *O problema do caixeiro viajante (PCV) é um conhecido problema de otimização combinatória. Sua versão mais simples consiste em determinar a menor rota para percorrer uma série de cidades (visitando-as uma única vez), retornando à cidade de origem. Ainda não foi possível provar que o problema possui solução ótima em tempo polinomial. Deste modo, observa-se uma tendência na utilização de métodos aproximados para resolve-lo. Este trabalho apresenta a aplicação de um desses métodos, conhecido como algoritmo genético (AG). O AG é uma meta-heurística da classe dos algoritmos evolutivos que usa técnicas inspiradas pela biologia evolutiva. Tais técnicas encontram boas aproximações e, por vezes, resultados ótimos para o PCV.*

1. Introdução

O problema do caixeiro viajante é um dos mais famosos no campo da otimização combinatória. Sua formulação é simples, é informado um conjunto de cidades e um custo $c_{i,j}$ associado a cada par i, j de cidades, o caixeiro deve iniciar iniciar a jornada em uma cidade inicial, passar por todas as demais cidades do conjunto apenas uma vez e retornar a cidade de origem. O problema consiste em encontrar a rota com menor custo associado total, ou seja, o menor caminho possível.

Apesar da simplicidade na descrição do problema, sua complexidade computacional aumenta exponencialmente a cada cidade inserida no conjunto de cidades. O PCV é classificado como um problema NP-Difícil, isto é, não é conhecido um método de se resolver através de um algoritmo polinomial (GAREY e JOHNSON, 1979). O interesse no estudo do problema vem das diversas aplicações da sua descrição em situações reais, algumas das mais variadas como:

- **Fabricação de placas de circuito eletrônico:** Na fabricação de uma placa de circuito eletrônico é necessário realizar furos de diferentes diâmetros e níveis. Para alterar o diâmetro do furo é necessário que a perfuradora troque a broca de perfuração e para realizar furos consecutivos ela deve mover a cabeça da máquina entre as posições dos furos. Isso consome tempo, então fica claro que o ideal é que a máquina realize todos os furos de um mesmo diâmetro, em uma sequência que poupe tempo, antes de trocar para a próxima broca e assim repetir o processo retornando a posição da máquina até a posição de origem para iniciar uma nova placa. As "cidades" são as posições iniciais e o conjunto de todos os furos que podem ser feitos com uma broca; a "distância entre as cidades" é o tempo gasto para mover a cabeça de uma posição para outra. A meta é minimizar o tempo de deslocamento da cabeça da máquina. Ainda, ao finalizar o processo de furos de uma placa, a peça perfuradora deve retomar ao ponto de partida, para iniciar a perfuração de uma nova placa.
- **Logística e Distribuição:** O PCV é amplamente utilizado em logística e distribuição para otimizar rotas de entrega, minimizando a distância percorrida por veículos e reduzindo os custos operacionais. Empresas de transporte, serviços de entrega e até mesmo aplicativos de navegação utilizam o PCV para determinar as melhores rotas a serem seguidas pelos veículos.
- **Design de antenas:** Em engenharia de comunicações, o problema do caixeiro viajante pode ser aplicado no projeto de antenas. O objetivo é determinar a localização mais eficiente para as antenas, levando em consideração fatores como a cobertura desejada e a minimização do custo de instalação.

Visto a gama de áreas em que as aplicações do PVC estão, existe um forte interesse no estudo de técnicas que entregam resultados admissíveis para a resolução do problema. Esses métodos são conhecidos como heurísticas e meta-heurísticas. Heurísticas encontram soluções sub-ótimas para pois aliam soluções de qualidade com baixo custo computacional. Além disso, apresentam bom desempenho, rapidez e simplicidade. As meta-heurísticas por sua vez podem ser conceituadas como a junção de métodos heurísticos para encontrar soluções de qualidade, aplicadas em problemas que não têm um algoritmo definido ou demandam um grande esforço computacional.

Nesse trabalho, utilizamos uma meta-heurística amplamente utilizada para encontrar soluções para o PCV, conhecida como Algoritmo Genético (AG). O AG é um método de otimização baseado em mecanismos evolucionistas, que seguem a ideia de seleção natural e da sobrevivência de seres mais aptos, amplamente estudada por Charles Darwin. Em AGs, uma população de possíveis soluções para o problema em questão evolui de acordo com operadores probabilísticos concebidos a partir de metáforas biológicas, de modo que há uma tendência de que, na média, os indivíduos representem soluções cada vez melhores à medida que o processo evolutivo continua (Tanomaru, 1995).

2. Referencial Teórico

Nessa secção apresentaremos a modelagem e teoria tanto do problema quanto do algoritmo. Para a aplicação do algoritmo no problema em questão algumas especificações e representações devem ser definidas. Na secção 2.1 definidos algumas propriedades do problema. Na secção 2.2 definimos etapas do AG. Na secção 2.3 são definidos os parâmetros utilizados para o AG.

2.1. PCV

A representação do caixeiro viajante utilizada nesse trabalho é a representação usual através de um grafo $G = (V, A)$, tal que V é o conjunto que representa os vértices do grafo, as cidades que devem ser visitadas, e A é o conjunto que representa as arestas, conexões entre as cidades.

Uma aresta $a \in A$ representa a conexão entre duas cidades i, j tal que $i \in V$ e $j \in V$. A ponderação da aresta é a distância entre as duas cidades. O grafo utilizado é um grafo completo de forma que $\forall(i, j)\ \exists a \in A$, isto é, existe caminho entre qualquer par de vértices. A distância entre as cidades é calculada por meio da distância euclidiana entre os pontos.

2.2. AG

A modelagem do algoritmo genético pode ser feita dividindo seu funcionamento em etapas. As etapas são os métodos bioinspirados que compõem o algoritmo genético.

2.2.1. População Inicial

A primeira etapa do algoritmo é a criação da população inicial. Em algoritmos genéticos a população inicial é composta de indivíduos que representam soluções do problema. No contexto do PCV, um individuo representa uma rota que visita todas as cidades (o retorno a cidade de origem será abordado durante a modelagem da função *fitness*). Uma população é composta por muitos indivíduos. O tamanho da população é um parâmetro crucial para o bom funcionamento do algoritmo.

Existem diversas formas de representar um indivíduo da população inicial para o PCV. Nesse trabalho utilizamos a representação *Path* que consiste em uma lista de cidades visitadas sem repetição.

Representação Path de um individuo: [0 5 4 11 8 6 7 1 3 2 9 10]

2.2.2. *Fitness*

Criada a população inicial é necessário avaliar o quão aptos são os indivíduos que compõe essa população. No PCV, a aptidão de um individuo pode ser definida utilizando a distância total da sua rota.

Dado um indivíduo $w = [a0\ a1\ a2\ a3\ a4\ a5\ a6\ a7\ a8\ a9\ a10\ a11]$ da população inicial, chamamos os elementos que compõem os indivíduos de genes. Podemos calcular a distância entre duas cidades calculando a distância euclidiana entre dois genes.

$$distancia(a0, a1) = \sqrt{(a0_x - a0_y)^2 + (a1_x - a1_y)^2}$$

Nesse cálculo $a0_x$ e $a0_y$ representam as coordenadas x e y no sistema cartesiano do ponto $a0$. Utilizamos o calculo da distância entre dois pontos para calcular o valor da rota. Esse cálculo deve respeitar a ordem dos genes e em seguida somar o valor da distância entre o último e o primeiro gene. Isso garante o retorno a cidade de origem.

$$rota(w) = \sum_{i=0}^{10} distancia(a_i, a_{i+1}) + distancia(a0, a11)$$

Com o valor da rota calculada, é possível definir a função *fitness* como o inverso do valor da rota total, isso garante que os melhores indivíduos, os que tem as menores rotas, terão um *fitness* maior e os indivíduos com maiores distancias *fitness* menores.

$$Fitness(x) = 1/rota(w)$$

2.2.3. Seleção

A seleção é um processo que visa escolher os indivíduos que irão realizar a reprodução de uma nova geração. Os operadores genéticos utilizados nessa etapa são: elitismo e roleta. O objetivo na combinação dessas duas técnicas é permitir que os indivíduos mais aptos, ou seja, aqueles que representam as melhores soluções para o problema, estejam na geração posterior mas também que exista um grau de aleatoriedade na combinação.

2.2.4. Elitismo

O elitismo se trata de um fator responsável por carregar o melhor cromossomo da geração *G* para a próxima geração *G+1*. Utilizamos esse fator para que a qualidade da melhor solução em cada geração aumente monotonicamente, ou seja, foi possível manter uma taxa de efetividade de geração para geração. A taxa de elitismo também é um fator fundamental no funcionamento do algoritmo, ela define a quantidade de indivíduos considerados mais aptos que irão compor a próxima geração.

2.2.5. Roleta

A seleção por roleta é uma técnica que atribui uma probabilidade de seleção a cada indivíduo da população com base em sua aptidão. Quanto maior a aptidão de um indivíduo, maior é a probabilidade de ser selecionado. Isso é feito através de uma roleta virtual, em que cada indivíduo possui um setor proporcional ao seu valor de aptidão. Em seguida, um número aleatório é gerado e a roleta é girada para determinar o indivíduo selecionado. Essa abordagem permite uma seleção mais diversificada, dando oportunidade para soluções promissoras com menor aptidão também serem selecionadas.

2.2.6. Reprodução

Na etapa de reprodução realizamos o processo de criação de novas soluções ou indivíduos a partir da combinação de informações genéticas (cada gene é uma cidade) presentes na população atual. Assim como na reprodução biológica, a reprodução no AG busca gerar descendentes que herdam características benéficas dos indivíduos existentes, visando aprimorar a qualidade das soluções ao longo das gerações. Os operadores genéticos utilizados nessa etapa são: *crossover* e mutação.

2.2.7. *Crossover*

Na etapa de *crossover*, também conhecida como "recombinação", ocorre a combinação dos cromossomos dos pais selecionados. O objetivo desse processo é criar novas soluções combinando características de duas soluções parentais existentes. Inicialmente, dois indivíduos (soluções) são selecionados com base em seu desempenho, utilizando algum critério de seleção (como a roleta). Em seguida, um ponto de corte é escolhido aleatoriamente na representação das soluções parentais. Os genes antes desse ponto de corte são copiados do primeiro pai para o filho resultante, enquanto os genes após o ponto de corte são copiados do segundo pai para o filho. Esse processo gera um novo indivíduo que combina informações de ambos os pais. No entanto, pode ocorrer a formação de cromossomos inválidos ou com duplicatas durante o *crossover*. Para resolver isso, é necessário realizar uma etapa de reparação ou de verificação da validade da solução resultante. O processo de *crossover* é repetido várias vezes, combinando diferentes pares de indivíduos, a fim de criar uma nova população de soluções potencialmente melhores para o problema do caixeiro viajante. Esse ciclo é realizado em conjunto com outros operadores genéticos (como a mutação) para melhorar gradualmente as soluções encontradas ao longo das gerações.

2.2.8. Mutação

A mutação apesar de simples é um recurso bastante poderoso na introdução da diversidade das gerações. O processo envolve a introdução de pequenas alterações aleatórias nas soluções geradas pelo *crossover*. São selecionados dois genes (cidades) aleatórios no cromossomo (solução) e esses genes trocam de posição na solução (como trata-se de um grafo completo no problema não há necessidade de se preocupar se há caminhos válidos). A mutação explora regiões do espaço de busca que podem não ser alcançadas apenas pelo *crossover*. Essa diversidade é importante para evitar a estagnação em soluções sub-ótimas e permitir a descoberta de novas soluções mais promissoras.

2.2.9. Critério de parada

O critério de parada foi definido como um número máximo de gerações (iterações do algoritmo). Nesse caso, o algoritmo continua a evoluir a população de soluções até atingir o limite de gerações estabelecido mesmo que não exista melhora nos indivíduos durante algumas gerações.

3. Metodologia

3.1. Pseudo-código

Os algoritmos genéticos são algoritmos de pesquisa heurística inspirados em processos evolutivos da natureza. Os algoritmos genéticos padrões são divididos em cinco passos:

1. Criar uma população inicial.
2. Calcular o *fitness*.
3. Selecionar os melhores genes.
4. Realizar o *crossover*.

5. Realizar a mutação.

Para o algoritmo do caixeiro viajante temos o seguinte algoritmo:

1. Criar uma população incial aleatoriamente.
2. Calcular o *fitness* do cromossomo.
3. Repita enquanto não finalizar:
   (a) Selecionar os pais.
   (b) Realizar o *crossover* e a mutação.
   (c) Calcular o *fitness* da nova população
   (d) Adicionar os novos genes como futuros progenitores.

Iremos demonstrar os passos do algoritmo na linguagem Python3 com o seguinte grafo como exemplo:

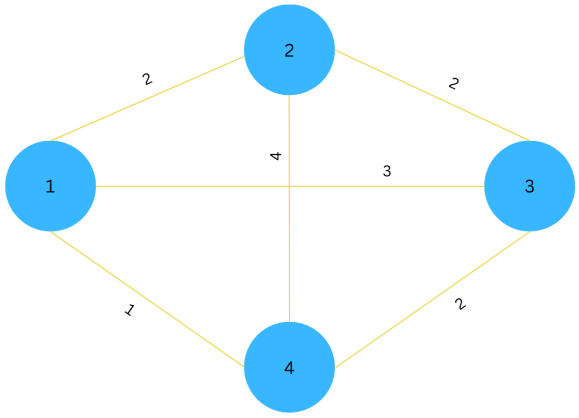

Figure 1. Grafo de exemplo

Para criar a população inicial:

Algorithm 1 generatePopulation(graph)

```
cities = [ ]
sizepop = Size of Population (parameter)
pop = [ ]
for x in range(len(graph)) do
    cities.append(x)
end for
for x in range(sizepop) do
    pop.append(rd.sample(cities, k = len(graph))
end for
```

1	2	3	4	1
1	4	3	2	1
1	3	4	2	1
1	3	2	4	1

Table 1. População gerada aleatoriamente

Para calcular o *fitness*:

Algorithm 2 routeTime(route, graph):

```
time = 0.0
for x in range(len(route) - 1) do
    if x > len(route) then
        time += graph[route[x]][route[x+1]]
    end if
end for
return performance
```

Cromossomo	*Fitness*
1 2 3 4 1	7 (1/7 = 0.143)
1 4 3 2 1	7 (1/7 = 0.143)
1 3 4 2 1	11 (1/11 = 0.091)
1 3 2 4 1	10 (1/10 = 0,1)

Table 2. Cálculo de fitness

A seleção dos pais introduz o elitismo e o método de seleção por roleta:

Algorithm 3 selection(pop, popFitness):

```
rankPop = rankRoutes(pop, popFitness)
selectedIndex = selectionIndex(rankPop)
selectedCromossomos = selectionCromo(pop, selectedIndex)
return selectedCromossomos
```

Continuando o exemplo, para um tamanho de elitismo igual a 2, teremos os seguintes genes selecionados:

1	2	3	4	1
1	4	3	2	1

Table 3. Genes com melhor performance

E utilizaremos o método da roleta para selecionar os outros dois que faltam para completar a população:

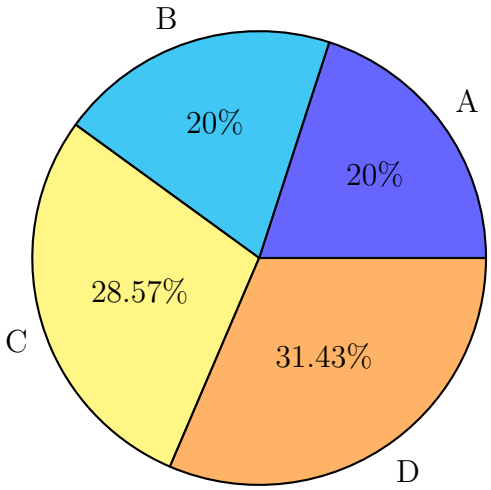

Figure 2. Representação da probabilidade dos genes serem escolhidos na roleta

Para prosseguir, vamos selecionar os números aleatoriamente na roleta, utilizaremos 58 e 29 no exemplo, que nos dão os genes B e C.

1	4	3	2	1
1	3	4	2	1

Table 4. Genes selecionados na roleta

Os dois indivíduos selecionados pelo elitismo avançam para a próxima geração mas ainda faltam dois indivíduos para completar a população. Portanto, em seguida acontece o *crossover*, cruzamento, aleatório entre todos os indivíduos selecionados pela roleta e pelo elitismo:

Algorithm 4 selectionIndex(rankPop):

```
selectionResults = [ ]
df = rouletteMethod()
selectionResults = elite(rankpop, eliteSize)
for i in range(0, len(rankpop) - eliteSize) do
    pick = 100 * rd.random()
end for
for i in range(0, len(rankpop)) do
    if pick <= df.iat[i,3] then
        selectionResults.append(rankpop[i][0])
        break
    end if
end for
return selectionResults
```

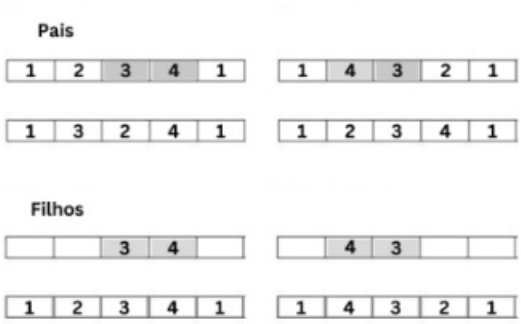

Figure 3. Como acontece o cruzamento

Por fim, pode acontecer a mutação dos genes. Como no problema do caixeiro viajante o indivíduo deve percorrer toda a população, a mutação acontece apenas trocando a ordem de visita das cidades, e, dependendo da modelagem, não pode alterar o último gene.

Algorithm 5 mutate(individual, mutationRate):

```
1: for gene1 do in range(len(individual)):
2:     if randomNumber < mutationRate then
3:         gene2 = int(rd.random() * len(individual))
4:         aux1 = individual[gene1]
5:         aux2 = individual[gene2]
6:         individual[gene1] = aux2
7:         individual[gene2] = aux1
8:     end if
9: end for
10: return individual
```

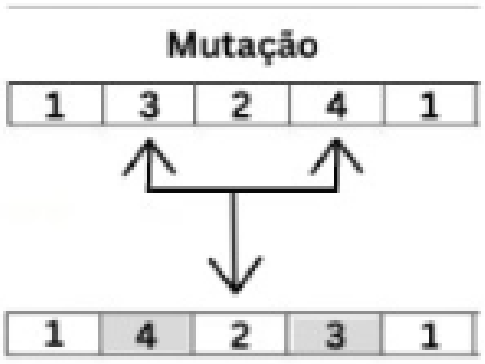

Figure 4. Como acontece a mutação

O critério de parada do algoritmo está sendo definido pelo número de gerações, executando o algoritmo enquanto não atingir o número de gerações desejado:

Algorithm 6 geneticAlgorithm(graphCities):

```
...
for i in range(0, numGenerations) do
    popFitness = fitness(pop, graphCities)
    selectedIndividuos = selection(pop, popFitness)
    popCrossover = reproduction(selectedIndividuos, int(len(selectedIndividuos) * 0.05))
    nextGeneration = mutation(popCrossover, 0.01)
    pop = nextGeneration
    progress.append(1/rankRoutes(pop, popFitness)[0][1])
end for
...
```

3.2. Experimentos

Os parâmetros utilizados durante os experimentos estão especificados na tabela 5. Esses parâmetros foram alcançados de forma empírica.

Entrada	Numero de cidades	População	Gerações	Elitismo (%)	Mutação (%)
Padrão	12	120	100	5	1
P01	15	225	250	5	1
WG22	22	440	500	10	1
ATT48	48	960	500	10	2

Table 5. Parâmetros utilizados na aplicação de cada entrada

A entrada padrão foi disponibilizada para o experimento base desse trabalho. A entrada PO1 é um conjunto de 15 cidades com custo conhecido igual a 291. A entrada WG22 descreve 22 cidades na Alemanha Ocidental. A entrada ATT48 é um conjunto de 48 cidades capitais dos Estados Unidos.

3.3. Análise dos resultados

Em execuções isoladas para cada entrada o algoritmo apresentou os seguintes resultados:

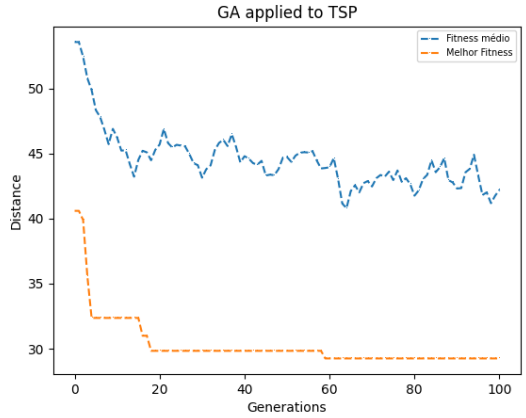

Figure 5. Gráfico da entrada Padrão

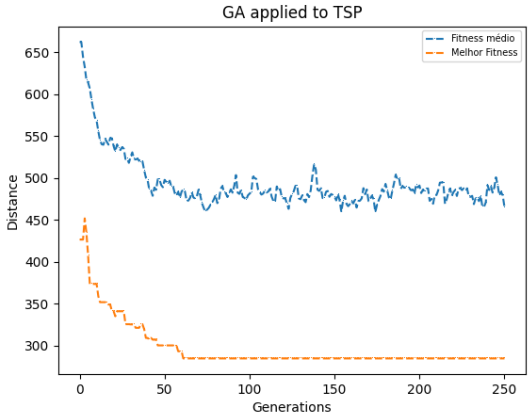

Figure 6. Gráfico da entrada PO1

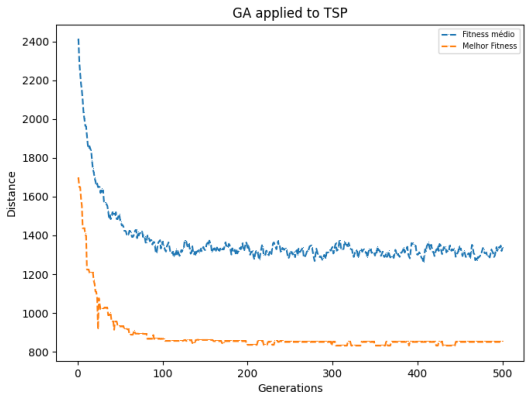

Figure 7. Gráfico da entrada WG22

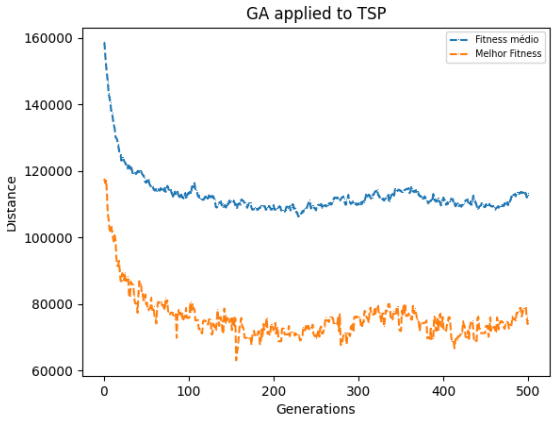

Figure 8. Gráfico da entrada ATT48

Nos gráficos podemos observar o desempenho do melhor individuo e o desempenho médio da população no decorrer das gerações. O padrão de convergência nas primeiras gerações está presente em todos os testes.

Na entrada Padrão obtivemos um valor ótimo próximo da geração de número 60 após várias gerações sem melhoria. O *fitness* médio dessa população caiu bastante nas primeiras 20 gerações e oscilou nas demais. Esses dados mostram que para essa entrada o algoritmo converge a um valor próximo do ótimo muito rápido.

Na entrada PO1, é possível notar uma convergência um pouco mais contínua do melhor individuo, seguida de um resultado constante nas gerações seguintes. Nos testes obtivemos um valor que superou o valor ótimo conhecido para entrada. Acreditamos que isso aconteceu devido devido a uma perda de precisão ao realizar cálculos com ponto flutuante na linguagem python.

A entrada WG22 apresentou grande convergência nas primeiras gerações seguida de pequenas oscilações nas gerações seguintes. O algoritmo não alcançou resultado ótimo, porém teve um resultado aproximado satisfatório.

A entrada ATT48 foi a que apresentou um maior desafio para o algoritmo. A

entrada segue a tendência de convergência muito rápida nas primeiras gerações. No entanto, nas gerações seguintes o resultado oscilou muito, alcançando seu melhor resultado entre as gerações 150 e 200 e apenas oscilando após isso. Essa oscilação revela uma necessidade de introduzir uma diversidade maior nas gerações para que o algoritmo continue convergindo.

3.4. Conclusão

Através da análise dos resultados obtidos e dos testes de execução realizados podemos observar alguns pontos:

- O AG apresenta convergência para um resultado aproximado de maneira muito muito eficiente logo nas primeiras gerações. Isso pode ser suficiente para a resolução de alguns problemas que não dependam de um resultado ótimo. Já na busca de um resultado ótimo é imprescindível estabelecer empiricamente um número de gerações adequado bem como garantir a introdução de diversidade nos cromossomos.
- A escolha dos parâmetros influenciam não só na qualidade do algoritmo quanto tempo de execução do programa. Uma taxa de mutação muito alta cria gerações muito estocásticas, não é possível ter controle dos genes mutados e isso pode ocasionar na perda de soluções promissoras. De maneira semelhante, um elitismo alto também não colabora, pois os cromossomos com os melhores genes passam a não ser diversos e isso pode levar a máximo local indesejado.
- O *Fitness* médio da população tem comportamento semelhante ao comportamento do *Fitness* do melhor individuo. Em geral, nas gerações que o *Fitness* do melhor individuo converge para uma distância menor, o *Fitness* médio também converge, isso demonstra que a população inteira convergiu para uma distância menor e não apenas o de melhor individuo.
- O AG demonstrou ser bastante flexível possibilitando a incorporação de diferentes heurísticas e técnicas de melhoramento, aumentando a qualidade das soluções encontradas.
- Apesar das qualidades do algoritmo, é importante ressaltar que mesmo utilizando um critério de parada que dependesse da estabilidade da distância percorrida pelo melhor indivíduo, isso não iria garantir que o resultado alcançado fosse ótimo, como os experimentos com a entrada WG22 mostraram o algoritmo pode alcançar um máximo local e não apresentar melhorias durante muitas gerações.

Portanto, o AG demonstrou ser uma abordagem eficaz em encontrar soluções ótimas ou aproximadamente ótimas para o problema, considerado um dos mais complexos na área de otimização combinatória. Os operadores genéticos também são ótima alternativas para lidar com a natureza combinatorial do TSP, permitindo a exploração de múltiplas soluções em paralelo e a evolução das melhores soluções ao longo das gerações.

References

GAREY, M. R.; JOHNSON, D. S. Computers and Intractability: a Guide to the Theory Of NP_Completness. 1. ed. New York: W. H. Freeman, 1979.

Tanomaru, J, Staff Scheduling by a Genetic Algorithms with Heuristic Operators. Chap 820. (1995).