Problema do Caixeiro Viajante com Algoritmo Genético

Clodoaldo B. da Fonseca, Jefferson Gabriel S. Mantovani, Pedro L. Perozin[1]

[1]Bacharelado em Ciência da Computação –
Universidade Tecnológica Federal do Paraná (UTFPR)
Campus Campo Mourão - Via Rosalina Maria Dos Santos, 1233 CEP 87301-899
Caixa Postal: 271 Campo Mourão - PR - Brasil

{clodoaldofonesca92, jmantomavani, pedroperozin}@gmail.com

Abstract. *mplementation of a genetic algorithm to solve the Problem Traveler Problem, a problem that has a simple and easy to understand, but is not an algorithm that solves the resolution.*

Resumo. *Implementação de um algoritmo genético para a resolução do problema do Caixeiro Viajante, um problema que possui uma formulação simples e de fácil compreensão, mas não há algoritmos eficientes que o resolvam.*

1. Introdução

Algoritmos genéticos são algoritmos inspirados e baseados na evolução e na genética. Eles fazem uma estratégia de busca paralela e estruturada, porém aleatória, que é voltada em direção ao reforço da busca de pontos de alta aptidão. Assim, os melhores resultados serão propagados para as próximas gerações.

Este conceito será usado para resolver o problema do caixeiro viajante, um problema de otimização combinatória mais estudados e conhecidos. Ao final, será comparado diversas abordagens do algoritmo genético para assim verificar as possíveis implicações no resultado final encontrado.

2. Objetivos

Este trabalho tem como objetivo, implementar um algoritmo genético para resolver o problema do caixeiro viajante e comparar quais mudanças no algoritmo resultam em um melhor resultado para o problema.

3. Problema do Caixeiro Viajante

O Problema do Caixeiro Viajante (TSP - Travelling Salesman Problem) é um dos problemas de otimização combinatória mais estudados e conhecidos. Ele é intrigante pois possui uma formulação simples e fácil de compreender, mas não existem algoritmos exatos eficientes que o resolvam. O problema de otimização é NP-difícil, enquanto o problema de decisão relacionado é NP-completo.[**?**]

Esse problema consiste em tentar determinar a menor rota para percorrer uma série de cidades, visitando uma única vez cada uma delas, e retornando à cidade de origem. Foi inspirado na necessidade dos vendedores em realizar entregas em diversos cidades percorrendo o menor caminho possível, reduzindo o tempo necessário para a viagem e os possíveis custos com transporte e combustível.

4. Algoritmo Genético

Um algoritmo genético é uma heurística de busca inspirada na teoria da evolução natural de Charles Darwin. Este algoritmo reflete o processo de seleção natural, onde os indivíduos mais aptos são selecionados para reprodução, a fim de produzir descendentes da próxima geração. [**?**]

O processo de seleção natural começa selecionando os indivíduos que tem o coeficiente de adaptação (fit) melhor. Eles vão produzir características que serão passadas para as gerações futuras. Se os pais tiverem um fit bom, seus filhos terão chances maiores de sobreviver. Esse processo é feito iterativamente e ao final, será encontrada a geração com os maiores fits. [**?**]

Sendo assim, 5 fases são consideradas no algoritmo genético:

1. População inicial: é o início do algoritmo, é onde são escolhidos os membros da população. Normalmente, a escolha é feita de maneira aleatória.

2. Função de aptidão (fitness): nessa fase, é calculado o aptidão (fitness) de cada membro da população.

3. Seleção: na seleção, escolheremos os pais que criaram a próxima geração. É levado em consideração o fitness, ou seja, os membros da população que possuem os maiores fitness, tem maiores chances de serem escolhidos.

4. Reprodução: é a criação da próxima geração. Escolhe-se um algoritmo para a reprodução.

5. Mutação: para evitar a estagnação, temos a mutação de um filho, ou seja, alteramos uma característica do filho gerado. Normalmente temos uma taxa de mutação para definir quais as probabilidades de acontecer a mesma.

4.1. População inicial

O processo começa com um conjunto de indivíduos chamado de População . Cada indivíduo é uma solução para o problema que você deseja resolver. Um indivíduo é caracterizado por um conjunto de parâmetros conhecidos como Genes . Os genes são unidos em um vetor com valores binários (0s e 1s) para formar um cromossomo.[**?**]

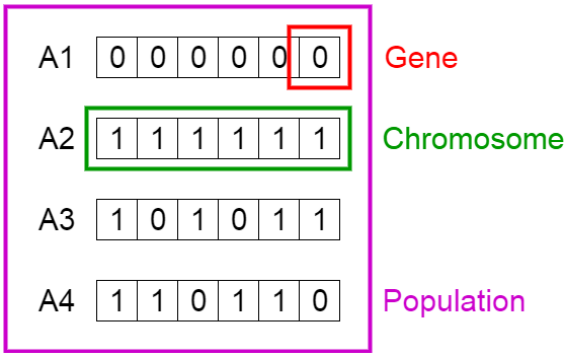

Figure 1. Gene, cromossomo e população

4.2. Função Fitness

A função fitness é a função que será o maximizada pelo algoritmo genético, neste caso é a soma total da distancia entre todas as cidades. A função recebe um vetor contendo a ordem de visita de todas os pontos e uma matriz contendo todas as distancias. A função retorna o custo de tomar esse determinado caminho, levando em conta o custo de retornar do ultimo ponto para o inicial, fechando assim o ciclo.

4.3. Reprodução

A reprodução, ou *crossover*, é a fase mais significativa em um algoritmo genético. Para cada par de pais a serem cruzados, um ponto de cruzamento é escolhido aleatoriamente dentro dos genes.

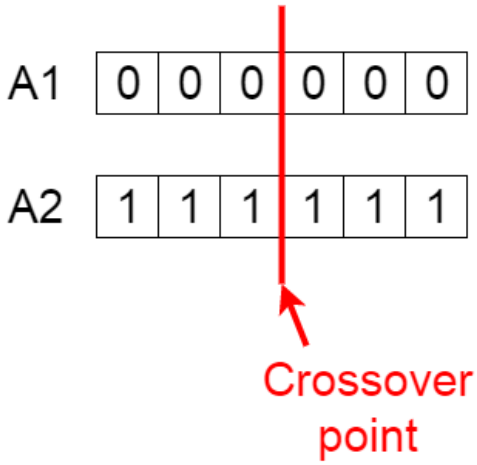

Figure 2. Crossover point

4.3.1. Crossover Ordenado

O *crossover* ordenado garante que a solução gerada continua sendo válida para o TSP, uma vez que todo nó de N aparece na rota uma única vez. Com o *crossover*, a reprodução acontece segundo o pseudo codigo a seguir: [**?**]

```
r = p1
p = lista da posicao de k elementos aleatorios em ordem
s = subconjunto com os k elementos de p1 correspondentes as posicoes na
p_ord = indices dos elementos de s ordenados em relacao a suas posicoes
para cada indice i dos elementos de s:
r[p[i]] = s[p_ord[i]]
```

4.3.2. Crossover Alternado

Além do *crossover* ordenado, também é comum usar o *crossover* alternativo. Este algoritmo combina as soluções *p1* e *p2* selecionando um ponto de corte. No entanto, diferentemente do *crossover* tradicional, cada elemento de *p2* é verificado antes de ser copiado para a solução, de forma que elementos já existentes não sejam duplicados. [**?**]

```
def crossover(self, p1, p2):
r = copy.copy(p1)
corte = random.randrange(1,len(p1))
for i in range(corte,len(p1)):
if(p1[i]not in p2[0:i]) and (p2[i] not in p1[0:i]):
p1[i] = p2[i]
return r
```

4.4. Mutação

Em alguns novos filhotes formados, alguns de seus genes podem ser submetidos a uma mutação com baixa probabilidade aleatória. Isto implica que alguns dos bits na cadeia de bits podem ser invertidos.[**?**]

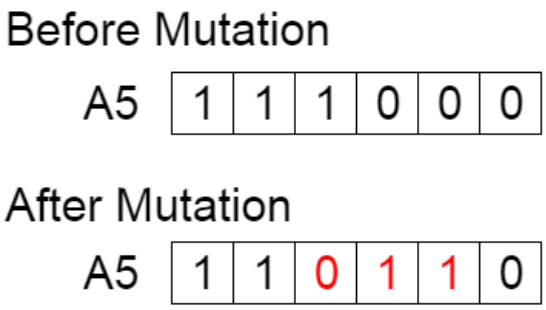

Figure 3. Crossover point

4.5. Algoritmo Genético

O algoritmo genético é a principal parte da implementação, ele consiste na simulação de um ambiente natural, onde os indivíduos se reproduzem e geram uma nova população que substitui a geração antiga. Para este algoritmo, cada individuo é uma possível solução do problema do caixeiro-viajante, sendo que as melhores soluções possuem maior chance de gerar novos filhos

4.6. Condição de Parada

O algoritmo termina se a população tiver convergido (não produz descendentes significativamente diferentes da geração anterior). Então, diz-se que o algoritmo genético forneceu um conjunto de soluções para o nosso problema.

A condição de parada implementada foi o numero de gerações sem ocorrer uma melhora no resultado obtido. Se não houver melhora, o algoritmo para.

4.7. Elitismo

Uma das estratégias usadas em algoritmos genéticos é denominada elitismo. Dada uma população com P indivíduos construída numa geração gi qualquer, estratégias elitistas mantém os k melhores indivíduos da geração $gk\text{-}1$. Desta forma, a busca tenta priorizar a recombinção das melhores soluções, garantindo sua seleção.[**?**]

Cada elemento dá população pai, é selecionado um elemento melhor na população filha, caso nenhum filho seja melhor que um determinado individuo pai, este individuo será mantido na população final. Caso não exista o elitismo, a população pai é totalmente substituída por seus filhos.

5. Avaliação Experimental

Afim de testarmos o desempenho do algoritmo implementado, utilizamos os seguintes problemas:

- a280
- berlin52
- kroA100

Todos os problemas listados estão disponíveis na biblioteca TSPLIB para uso acadêmico.

Na figura abaixo podemos observar os resultados, onde o Elitismo não ocorreu, observa-se que não há melhora constante, pois não ocorreu a seleção dos indivíduos, tendo ainda um tempo de execução menor dada a condição de parada, logo, não tendo muitas gerações. Afim de melhorar esse resultado pode-se aumentar o valor da condição de parada, consequentemente aumentando o numero de gerações e o tempo de execução.

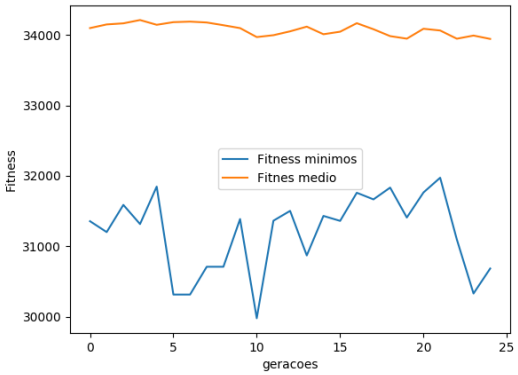

Figure 4. Não elitismo

Já aqui observamos uma melhora constante no resultado já que utilizamos o elitismo para que seja escolhido os melhores indivíduos da população para a próxima geração.

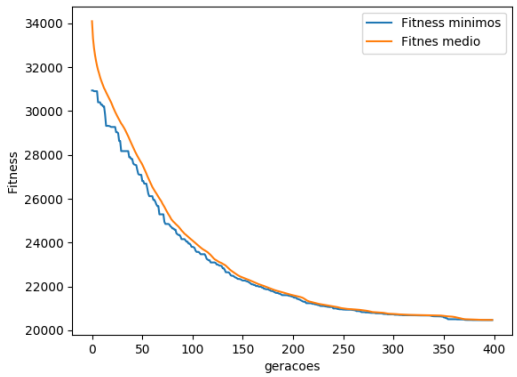

Figure 5. Elitismo

6. Conclusões

Conclui-se que é factível a utilização o algoritmo genético para a solução de um problema real. Observa-se que mudanças aparentemente pequenas tem impacto direto no desempenho e resultado do algoritmo, como também, a utilização ou não do elitismo pode impactar diretamente resultado.

7. References

MALLAWAARACHCHI, Vijini. Introduction to Genetic Algorithms: Including Example Code. 2017. 2017. Disponível em: <https://towardsdatascience.com/introduction-to-genetic-algorithms-including-example-code-e396e98d8bf3>. Acesso em: 29 out. 2018.

FOLEISS, J. H. Trabalho 1: Problema do Caixeiro Viajante com Algoritmo Genético. [S.l.], 2018. Especificação do Trabalho 1.

TRAVELLING salesman problem. Wikimedia Foundation, 2018. Disponível em: <https://en.wikipedia.org/wiki/Travelling salesman problem>.

APPLEGATE, David L. [et al.] – The travelling salesman problem: a computational Study. Princeton: Princeton University Press, 2006. ISBN 978-0-691-12993-8