

UNIVERSIDADE DO SUL DE SANTA CATARINA

JONATHAN NEVES

Algoritmo Genético:

Desafio do Caixeiro Viajante

TUBARÃO

2020

 **Universidade do Sul de Santa Catarina – UNISUL**

**Curso de Ciência da Computação**

**Disciplina:** MODELOS EVOLUCIONÁRIOS E TRATAMENTO DE INCERTEZAS

**Professor:** Clávison M. Zapelini **E-mail:** [clavison.zapelini@unisul.br](mailto:clavison.zapelini@unisul.br)

**Semestre:** 2020-A

**AVALIAÇÃO PRÁTICA SOBRE ALGORITMOS GENÉTICOS**

**Problema do Caixeiro Viajante**

O problema do Caixeiro Viajante é encontrar o melhor caminho a ser percorrido, no menor tempo, levando em consideração a tolerância de erros e a quantidade de cidades, nas quais não podem ser repetidas, isto é, se o número de cidades for igual a dez, o caixeiro deve passar pelas dez cidades, sem retornar a nenhuma, no menor tempo possível (Ferreira, 2007).

Para Reinelt (1994) o problema do caixeiro viajante atrai pesquisadores de diversas áreas, como por exemplo, físicos, biólogos, matemáticos, entre outros, já que se utilizando dessa técnica, é possível encontrar várias respostas para questões de otimização combinatória e pela simplicidade de sua formulação. Goldberg & Luna (1999) apud Cunha, Bonasser & Abrahão (2002) a origem do problema é um jogo criado por William Rowan Hamilton, com objetivo de traçar um roteiro através dos vértices de um dodecaedro que iniciam e terminam no mesmo vértice (cidade) sem repetir visitas

**AVALIAÇÃO**

Desenvolver um sistema, com interface gráfica, que possibilite cadastrar uma quantidade de cidades (LIMITAR A 10) com os respectivos nomes e os tempos (em minutos) entre um e outra.

Por exemplo, vamos supor que o usuário tenha solicitado o cadastro de 5 cidades:

A – B – C – D – E

O cadastro dos tempos deve prever que seja possível estabelecer o tempo entre todas as opções:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | **A** | **B** | **C** | **D** | **E** |
| **A** | 0 | 10 | 15 | 5 | 12 |
| **B** | 10 | 0 | 70 | 52 | 27 |
| **C** | 15 | 70 | 0 | 120 | 14 |
| **D** | 5 | 52 | 120 | 0 | 38 |
| **E** | 12 | 27 | 14 | 38 | 0 |

Utilizar algoritmos genéticos para estabelecer a melhor rota de modo a atender ao problema do Caixeiro Viajante. Ele deve percorrer todas as cidades, sem repetir nenhuma delas e voltar para a cidade de origem, com o menor tempo possível.

# Qual forma de representação dos indivíduos foi utilizada para representar cada solução.

Foi utilizado para representar os indivíduos a permutação, onde cada cidade receberá uma Letra em ordem alfabética de cadastro.

**Ex.: A->D->B->C.**

Foi escolhido permutação, pois ela é a melhor forma de representar uma ordem específica de tarefas como é o caso do caixeiro viajante, onde devem passar por todas as cidades sem repeti-las.

# Como foi feito o cálculo do *fitness* de cada possível solução

Para realizar o cálculo do fitness foi feito a soma dos tempos entre a sequência das cidades. Lembrando que também foi necessário realizar a soma do tempo entre a cidade final com a cidade inicial. Pois o caixeiro passa por todas as cidades e depois retorna para cidade inicial.

Ex.: A para B = 30min; A para C = 15 min; B para C = 5min;

A->C->B->A = [*tempoAC+tempoCB+tempoBA]* = 50min;

# Qual percentual de mutação genética (caso tenha sido utilizada)

Não foi utilizado nenhuma taxa de mutação pois foi realizado a mutação selecionando 2 cidades aleatórias do pior indivíduo da geração e trocado de lugar entre si.

Ex.: [A->**C**->B->**D**] => [A->**D**->B->**C**]

Este é o melhor método para utilizar em permutação para evitar casos inválidos, onde o indivíduo possui um caminho com cidades repetidas.

# Como se deu o método de seleção dos indivíduos para reprodução no momento de estabelecer as próximas gerações

Os 4 indivíduos são avaliados com base no seu fitness e são escolhidos e mantidos os 2 indivíduos com menor tempo (*o mais apto*) entre os 4 da geração atual para gerar descendentes para a próxima população. Logo o objetivo do problema é encontrar a sequência de cidades com o menor tempo.

O algoritmo inicia a geração com 2 indivíduos, e gera mais dois indivíduos através do crossover, e por fim realiza a mutação no pior indivíduo, finalizando com 4 indivíduos. No final é escolhido apenas os 2 indivíduos mais apto que serão mantidos para a próxima geração e repete o processo até cumprir o critério de parada.

Para o crossover foi utilizado *Non-Wrapping Ordered Crossover (NWOX).* Consiste em pegar uma posição inicial e uma posição final, e adicionar todos os valores presente no pai entre estas posições ao filho. Após isso, é preciso completar o indivíduo com as cidades restantes, então é realizado uma verificação se o filho 1 não contém a cidade do pai 2, ou vice-versa, então é adicionado ao indivíduo.

***Posição inicial***= 2; ***Posição Final*** = 4;

Ex.: Pai 1 [A->**B->C->D**->E] >> Filho 1[**B->C->D**->\_->\_] >> Filho 1[B->C->D->**E->A**]

Pai 2 [E->**D->C->B**->A] >> Filho 2[**D->C->B**->\_->\_] >> Filho 2[D->C->B->**A->E**]

# Qual o critério de parada.

O critério de parada é a escolha do usuário, por gerações ou automático:

* Geração: O usuário pode escolher o número de gerações desejadas, ou seja, o algoritmo irá executar *n* vezes.
* Automático: O algoritmo sempre verifica a cada geração se ele encontrou uma solução melhor que a atual, quando encontrado ele reinicia o contador (nº de gerações após encontrar a melhor solução) e vai incrementando cada vez que não encontra nenhuma solução melhor. Quando o contador for igual a 50 e o algoritmo não achar nenhuma solução melhor, o algoritmo para. (Pode acontecer de não encontrar a melhor solução durante este intervalo).

# Fluxograma do algoritmo.

