Universidade Federal de Pernambuco

Departamento e Ciência da Computação

Pós-Graduação em Ciência da Computação

Especificações Projeto Algoritmos 1: BRKGA aplicado com Problema do Caixeiro Viajante – PCV

Amanda da Silva Xavier

Andersson Alves da Silva

Prof.: Fernando Castor

Recife

2020

1 Introdução

O relatório refere-se ao projeto da disciplina Algoritmos I como parte da nota para o conceito final.

Uma das motivações para a escolha do tema a ser abordado neste projeto foi oriunda de um trabalho que está sendo desenvolvido com o professor Ricardo Martins (orientador de ambos alunos). O trabalho é composto por duas etapas em que a primeira delas consiste na compreensão da meta-heurística BRKGA (*Biased Random-Key Genetic Algorithm*).

Uma outra motivação consistiu no quão novo é este assunto, uma vez que não se encontrou nada desse algoritmo implementado em *Python* disponibilizado na literatura. Dado isso, implementamos o BRKGA utilizando linguagem de programação Python a partir dos conhecimentos adquiridos na disciplina de Algoritmos I para o projeto final. Isso contribuiu para o melhor entendimento sobre o algoritmo e com o desenvolvimento da primeira parte do projeto envolvido com o professor Ricardo.

2 BRKGA

O Algoritmo Genético com Chaves Aleatórias Tendenciosas (*Biased Random-Key Genetic Algorithm* - BRKGA) é uma variação do famoso Algoritmo Genético (Genetic Algorithm - GA). O GA utiliza o conceito de sobrevivência do mais apto para encontrar soluções ótimas ou aproximadas, envolvendo a analogia entre indivíduos em uma população e uma solução.

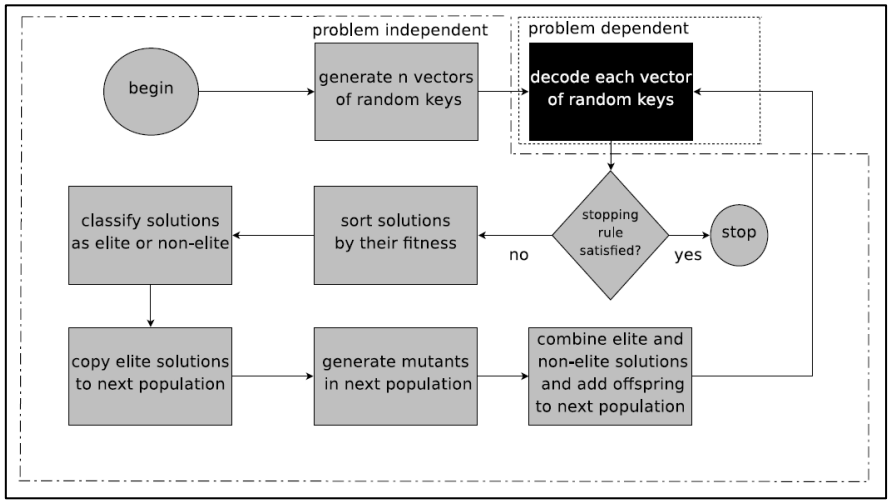
O GA possui certas características particulares, que precisam ser esclarecidas:

- Cada indivíduo tem um **cromossomo** correspondente que codifica a solução;
- Um **cromossomo** consiste em uma sequência de **genes**;
- Cada **gene** pode assumir um valor chamado de **alelo**, de algum alfabeto;
- Um cromossomo associa ao gene uma **aptidão** (*fitness*) correlacionado ao valor de uma Função Objetivo, correspondente da solução que codifica.
- Os GA's desenvolvem um conjunto de indivíduos que compõem uma **população** ao longo de várias **gerações**;
- A cada **geração**, uma nova população é criada combinando elementos da população atual para produzir os **descendentes** (*offspring*) que compõem a nova geração.

- A **mutação** aleatória também ocorre em algoritmos genéticos como um meio de escapar do aprisionamento em mínimos locais.
- O conceito de **sobrevivência** do mais apto usado pelo GA refere-se ao processo dos indivíduos serem selecionados para **acasalar** (*mating*) e produzir os descendentes (*offspring*) por meio de um processo chamado **cruzamento** (*crossover*).
- Os indivíduos são selecionados **aleatoriamente**, mas aqueles com melhor aptidão (*fitness*) são preferidos aos que são menos aptos.

O BRKGA também parte deste princípio e considera que cada **cromossomo** é representado como uma sequência ou vetor de números reais gerados aleatoriamente entre [0, 1]. A principal característica do BRKGA é que cada elemento é gerado combinando um elemento selecionado aleatoriamente da partição chamada **Elite** (os pais mais aptos) na atual população e um elemento da partição **Não Elite** (os pais que não são os mais aptos) [1]. O BRKGA prioriza a escolha dos pais da Elite para o processo de cruzamento (Crossover) da geração seguinte [2]. A Figura 1 apresenta o fluxo do BRKGA considerado no programa que implementado neste projeto.

Figura 1 – Fluxograma do BRKGA



Fonte: [1]

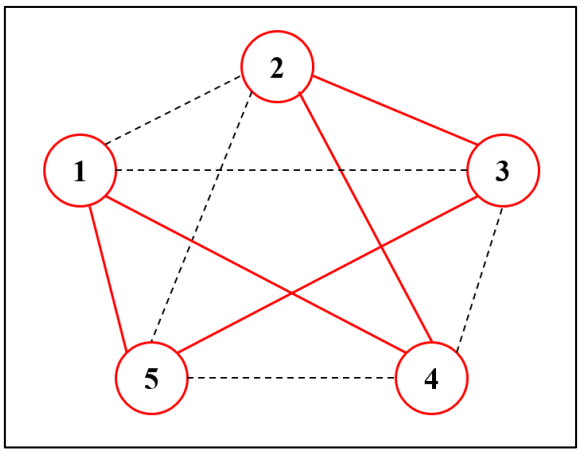
Dentro do BRKGA, inicialmente são gerados os indivíduos por meio de um vetor de chaves aleatórias cada, para compor a população inicial. Em seguida, o decoder interpreta cada vetor de acordo com a função objetivo desejada e calcula o fitness para cada indivíduo. Se um critério de parada for satisfeito, então tem-se a melhor resposta, caso contrário, cada indivíduo é ordenado segundo o seu fitness. A ordem dos indivíduos os classifica como elite ou não-elite. Esse critério é definido pelo parâmetro p_e que representa a proporção de indivíduos elite dentro da população total p. Da mesma forma,

a quantidade de indivíduos não-elite considerada é representada por($p - p_e$). Todos os indivíduos elite são copiados para compor a próxima geração, enquanto que os demais indivíduos são compostos por uma parcela de mutantes p_m gerados aleatoriamente e a quantidade de descendentes ($p - p_e - p_m$) gerados no processo de *crossover*, onde se escolhe aleatoriamente um indivíduo da população elite e um indivíduo da população não-elite para o acasalamento. Nesta etapa, é gerado um vetor de números aleatórios entre [0, 1] que serão confrontados por um parâmetro ρ, que representa a probabilidade do descendente herdar o alelo do pai elite, onde $\rho > 0.5$. Após criado a nova população, o procedimento repete com o *decoder* interpretando a informação para a função objetivo do problema.

3 Problema do Caixeiro-Viajante

A validação do modelo proposto foi feita a partir da implementação do BRKGA para resolver uma aplicação do Problema do Caixeiro-Viajante (PCV), um famoso problema combinatório, que pode ser resumido a partir de dado um conjunto com *N* locais e o custo da viagem (distância) entre cada par deles, determina-se o menor percurso de modo a visitar todos os locais e retorne ao seu ponto de origem, visitando cada local exatamente uma vez [3]. O problema pode ser representado por meio de um grafo, onde os nós representam os locais e as arestas representam a distância entre os locais. A Figura 2 demonstra essa representação.

Figura 2 – Representação do PVC em forma de grafo



Fonte: Autores

O PCV é chamado simétrico quando a distância entre dois nós (locais) quaisquer *i* e *j* independe do sentido, isto é, quando $d_{ij} = d_{ji}$; caso contrário o problema é

denominado assimétrico [4]. Sob a ótica de otimização, o PCV pertence à categoria conhecida como NP-difícil (do inglês "*NP-hard*"), o que significa que possui ordem de complexidade exponencial. Em outras palavras, o esforço computacional para a sua resolução cresce exponencialmente com o tamanho do problema (dado pelo número de pontos a serem atendidos) [4]. Portanto, diversos pesquisadores descrevem abordagens heurísticas ou aproximativas para resolução do PCV, de modo a reduzir esse esforço computacional.

Uma alternativa é o uso do BRKGA para sua resolução, o que foi utilizado neste trabalho para validar a implementação do algoritmo.

3.1 Aplicação do BRKGA para resolução do PCV

O BRKGA recebe como entrada o número N de cidades/locais a serem visitados e cada cidade possui um par ordenado (X, Y) referente a suas coordenadas no plano cartesiano. A distância entre duas cidades pode ser calculada pela distância Euclidiana, apresentado na equação 01.

$$Distância\ Euclidiana\ = \sqrt{(X_b - X_a)^2 + (Y_b - Y_a)^2} \quad (01)$$

O BRKGA é responsável por tratar as rotas viáveis para cada indivíduo da população total e para cada etapa foi computado o fitness de cada rota por meio da função objetivo que minimiza a distância total percorrida, conforme equação 02. Onde d_{ij} representa a distância entre as cidades i e j e X_{ij} é um número binário que assume 1 se a cidade *i* for visitada antes da cidade *j*, ou 0, caso contrário.

$$Função\ Objetivo\ = d_{ij}X_{ij} \quad (02)$$

O procedimento no BRKGA é rodado até que um total de *G* gerações da população sejam completadas. A saída do BRKGA será uma rota viável com o melhor fitness, ou seja, com a menor distância percorrida para atender todas as cidades.

4 Bibliotecas

Dentro todos os conhecimentos adquiridos na disciplina, a grande maioria foi considerado no trabalho: funções, listas, dicionários, tipos de variáveis e estrutura lógica.

Além disso, outros conhecimentos acerca da linguagem Python foram necessários a fim de facilitar a implementação do código, como também melhorar o desempenho do algoritmo.

4.1 Numpy

Numpy[1] é uma biblioteca poderosa para computação científica, que é usada para realizar cálculos em arrays multidimensionais de alto desempenho. Ele fornece um grande conjunto de funções e operações que ajudam os programadores a executar facilmente cálculos numéricos. Uma matriz numpy é uma grade de valores, todos do mesmo tipo, e é indexada por uma tupla de inteiros não negativos. No programa, o Numpy foi utilizado para criar, manipular e organizar as matrizes arrays com os indivíduos dentro da população.

4.2 Pandas

O Pandas[2] é uma biblioteca que fornece estruturas e ferramentas de análise de dados de alta performance por meio de *dataframes*. Os *dataframes* são um tipo de planilha, como no Microsoft Excel, composto por linhas e colunas, e que possuem diversas funcionalidades para tratamento e análise de dados, como agregação e contagens.

Como foi utilizado no projeto: utilizado para importar os dados de entrada para o Problema do Caixeiro Viajante a partir de uma planilha em formato ".csv" e criar os pontos no espaço bidimensional das cidades.

4.3 Math

A biblioteca Math[3] é usada quando se necessita ter acesso as funções matemáticas, como funções de potência, logarítmicas e trigonométricas. Ela foi utilizada no programa para gerar a raiz quadrada na determinação da distância Euclidiana, conforme equação 01.

[1] https://numpy.org/doc/stable/user/basics.html
[2] https://pandas.pydata.org/pandas-docs/stable/getting_started/intro_tutorials/index.html
[3] https://docs.python.org/3/library/math.html

4.4 Random

O Random[4] é responsável por implementar geradores de números pseudoaleatórios para várias distribuições. Ela se torna ideal para gerar números aleatórios ou selecionar um item aleatoriamente a partir de um conjunto de dados. O Random foi utilizado no projeto para gerar a população inicial de forma aleatória com números reais entre 0 e 1, para escolher aleatoriamente indivíduos para o crossover e e criar mutantes aleatórios na etapa de mutação.

4.5 Matplotlib.pyplot

O Matplotlib[5] é muito útil quando o objetivo é gerar imagens que representem os dados, sendo apresentados em diversas formas de gráficos a partir de um conjunto de dados selecionados. O "pyplot" possui uma coleção de funções que fazem o matplotlib funcionar como um MATLAB. Cada função pyplot faz alguma alteração em uma figura: por exemplo, criando uma figura, criando uma área de plotagem em uma figura, plotando algumas linhas em uma área de plotagem, decorando a plotagem com rótulos, etc.

No programa o matplotlib.pyplot foi utilizado para criar um gráfico do plano cartesiano representando os nós (cidades) e as arestas (caminhos).

5 Resultados

Para o trabalho foram considerados um conjunto de N cidades representadas por um dicionário, no qual cada *key* recebe um rótulo para a cidade e o seu *value* representa as coordenadas X e Y em um plano cartesiano, conforme apresentado abaixo.

Foi utilizado 4 instâncias em arquivo ".csv" contendo 25, 35, 50 e 100 cidades para encontrar a menor rota possível de acordo com os seguintes parâmetros.

Seguem os resultados obtidos pelo BRKGA para cada instância.

N = 25 cidades

Geração: 100 gerações
População: 10 indivíduos
Fitness (Função objetivo) = 8.489,411885597608

[4] https://docs.python.org/3/library/random.html

[5] https://matplotlib.org/tutorials/index.html

[1, 23, 15, 14, 16, 19, 9, 17, 3, 12, 5, 22, 11, 13, 7, 2, 24, 18, 0, 8, 10, 20, 21, 6, 4, 1]

N = 35 cidades

Geração: 100 gerações
População: 100 indivíduos
Fitness (Função objetivo) = 8.389,251793810994
Rota = [0, 1, 3, 5, 2, 7, 8, 9, 4, 6, 11, 12, 10, 15, 16, 13, 14, 17, 18, 19, 20, 21, 22, 25, 23, 24, 28, 26, 27, 29, 30, 31, 32, 34, 33, 0]

N = 50 cidades

Geração: 100 gerações
População = 100 indivíduos
Fitness (Função objetivo) = 11.273,894688448092
Rota = [0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 12, 11, 13, 14, 16, 15, 17, 18, 21, 19, 20, 23, 22, 24, 27, 25, 26, 28, 29, 30, 31, 32, 33, 34, 35, 36, 37, 38, 40, 39, 41, 42, 44, 43, 48, 45, 46, 47, 49, 0]

N = 100 cidades

Geração: 100 gerações
População: 100 indivíduos
Fitness (Função objetivo) = 38.192,45646287143
Rota = [22, 39, 14, 85, 46, 92, 58, 49, 66, 35, 9, 23, 19, 38, 45, 7, 83, 28, 30, 70, 73, 57, 62, 13, 41, 36, 4, 78, 24, 54, 68, 69, 76, 18, 33, 84, 8, 25, 56, 52, 16, 44, 91, 65, 99, 63, 40, 32, 42, 10, 87, 20, 27, 53, 48, 75, 60, 59, 61, 55, 6, 96, 89, 2, 43, 90, 51, 64, 71, 26, 34, 80, 5, 88, 12, 79, 98, 11, 47, 29, 37, 3, 94, 50, 67, 86, 82, 74, 77, 15, 17, 95, 31, 81, 72, 21, 97, 0, 1, 93, 22]

6 Considerações Finais

De acordo com os resultados, mesmo para instâncias consideradas médias como 100 cidades, o algoritmo foi rápido em sua resolução e ainda conteve ótimos resultados, procurando minimizar a distância total de cada percurso.

Para melhorar a visualização dos resultados obtidos pelo fitness na função objetivo pensa-se em realizar visualização de gráficos de linha para verificar a cada geração como o fitness se comportou.

7 Referências

[1] GONÇALVES, J. F.; RESENDE, M. G. C. Biased random-key genetic algorithms for combinatorial optimization. *Journal of Heuristics*, v. 17, n. 5, p. 487-525, 2011.

[2] GONÇALVES, J. F.; DE ALMEIDA, J. R. A hybrid genetic algorithm for assembly line balancing. *Journal of Heuristics*, v. 8, n. 6, p. 629-642, 2002.

[3] GUEDES, A. C. B.; LEITE, J. N. F.; ALOISE, D. J. Um algoritmo genético com infecção viral para o problema do caixeiro viajante. *Revista PublICa*, v. 1, n. 1, 2005.

[4] CUNHA, C. B., BONASSER, U. O.; ABRAHÃO, F. T. M. Experimentos computacionais com heurísticas de melhorias para o problema do caixeiro viajante. In: *Anais do XVI Congresso de Pesquisa e Ensino em Transportes,* ANPET, Natal-RN, v. 2, p. 105-117, 2002.