Resolução do Problema do Caixeiro viajante com Algoritmo de Distribuição Estimada

Fernando Veras, Gleidson Mendes

April 11, 2016

Abstract

This paper reports the development and performance of an aplication to solve instances of the Traveling Salesman Problem. The aplication uses the EDA (Estimation of Distribution Algorithm) aproach on optimization and a part of the classes and methods provided by the inspyred library.

1 Introdução

O problema do caixeiro viajante é um problema NP-Completo amplamente estudado por conta de sua aplicação para resolução de problemas reais. Dentre os frameworks disponíveis que implementam (meta)heurísticas de otimização escolhemos o inspyred, que possui uma série de algoritmos inspirados na biologia, dentre eles colônia de formigas e enxame de partículas. A heurística escolhida porém foi a da Estimativa de Distribuição, que é um algoritmo evolutivo, porém usa modelos probabilísticos para a geração de novas populações. No trabalho a seguir deseja-se mostrar o processo de desenvolvimento do programa para a resolução do problema do caixeiro viajante utilizando EDA. O trabalho estrutura-se da seguinte forma:
No tópico 3 é apresentada a fundamentação teórica, mostrando o problema do caixeiro viajante e algoritmos utilizados para resolvê-lo. No tópico seguinte mostra-se os métodos utilizados, neste tópico apresentamos a meta heurística escolhida e como ela funciona para o caixeiro viajante e discutimos sobre como devemos tratar certas fases do desenvolvimento do algoritmo assim como a alteração do problema do caixeiro viajante para a utilização do caixeiro viajante Lobato.
No tópico 5 temos a apresentação dos resultados (por gráficos e análise) obtidos com a execução do programa em diferentes tipos de arquivos de entradas. E por fim a conclusão.

2 Objetivo

O objetivo do trabalho produzir um algoritmo capaz de produzir soluções para instâncias do Problema do Caixeiro Viajante utilizando: uma meta-heurística proposta em um artigo ou trabalho semelhante; e um framework para algoritmos de otimização. Então, testar o desempenho do algoritmo produzido quando aplicado na resolução de exemplos dados do problema do caixeiro viajante. E, por fim, apresentar uma análise dos resultados obtidos.

3 Fundamentação Teórica

3.1 Problema do Caixeiro Viajante

O Problema do Caixeiro Viajante (TSP) é a abstração de um problema de descoberta de uma determinada rota. No TSP, o agente em questão necessita partir de uma cidade de origem, passar por todas as N cidades apenas uma única vez e voltar a origem, percorrendo o menor caminho possível.
A complexidade deste problema é fatorial - n!, sendo N o número de cidades. O artigo Robles et al. (2002) explana melhor sobre a origem do problema do caixeiro viajante.

3.2 Algoritmos para O problema do caixeiro viajante

O artigo citado acima, apresenta primeiramente uma revisão sobre algoritmos das heurísticas existentes, que podem ser separados em 2 formas:

1. Heurísticas construtivas (Como vizinho mais próximo, guloso) e heurísticas aprimoradas (2 opt, 3 opt).

2. Meta-heurísticas, que se baseiam na existência de uma população inicial (soluções iniciais), que são selecionadas e então sofrem mutações e então realocadas na população resultado num conjunto de soluções (Algoritmo Genético, Programação evolutiva).

3.3 Algoritmo EDA

Após essa introdução o artigo apresenta a meta-heurística escolhida: o EDA (Estimation Distribution Algorithm). EDA é um método de otimização estocástica que guiam a busca para o ótimo atravéz da construção de um modelo probabilístico de soluções candidatas promissoras. A otimização é vista como uma série de melhoramentos gradativos.
Pertence a classe de algoritmos evolutivos, mas diferencia-se dos demais algoritmos de por gerar novos candidatos por probabilidade de distribuição codificada por distribuições bayesianas ou gaussianas.
O funcionamendo do EDA será explicado na seção seguinte, juntamente com o modo como foi implementado.

3.4 Framework

Depois de pesquisas sobre qual framework estavam disponíveis com suporte de programação da meta heurística escolhida neste trabalho. Foi decidido a utilização do Framework Inspyred (Aaron Garrett, a) e o código fonte da framework pode ser encontrado em (Aaron Garrett, b) esse framework foi implementado na linguagem de programação Python (Python Software Foundation).
O framework é totalmente modulado, permitindo a alteração de todas as funcionalidades de acordo com a necessidade do usuário. Assim, a framework torna-se totalmente adaptável, não somente para a meta heurísitica apresentada neste trabalho, mas também como várias outras.

4 Descrição da Proposta

4.1 Meta-heurística

Na figura 1 temos a apresentação do pseudo código de EDA

Step 1: Generate M individuals (the initial population) randomly
Step 2: Repeat until a stopping criterion is met
Step 2.1: Select s individuals, $s \leq M$ from the population, according to a selection method
Step 2.2: Estimate the probability distribution of an individual being among the selected individuals
Step 2.3: Sample M individuals (the new population) from the probability distribution found before

---

Figure 1: Pseudo Código EDA

No passo 1, temos a geração de uma população inicial, ou seja, temos a criação de uma quantidade X de indivíduos aleatórios, que no caso são permutações das cidades do problema. Dentro da permutação é importante que uma cidade não deve aparecer 2 vezes na permutação e nem aparecer nenhuma vez, para que tal indivíduo seja válido (Quando mencionado válido, neste artigo, é referido ao fato da permutação seguindo as regras apresentadas anteriormente. Após essa validação, todas as soluções são candidatas). No segundo passo temos um loop que vai existir enquanto um critério de parada seja não for alcançado. Dentro deste segundo passo, teremos sub passos nos quais, o primeiro irá selecionar uma quantidade de indivíduos da população existente, em que essa quantidade tem que ser menor ou igual ao número de indivíduos da população. No segundo passo, é feita a estimativa da probabilidade de distribuição de um indivíduo estar entre os indivíduos escolhidos. E por último esses novos indivíduos são recolocados na posição existente, como um todo, ou por partes.

4.2 EDA com valores reais

EDA consegue trabalhar tanto com valores inteiros quanto com valores contínuos, mas os processos de tratamento diferem entre si. O método de distribuição, transformação da distribuição e o cálculo do fitness. Neste trabalho estaremos executando EDA com valores reais
EDA com valores inteiros não necessita desta transformação, pois os valores que possui, já estão contidos dentro do valor das cidades. Entretanto, para uma distribuição real não é possível saber a qual cidade o valor em questão de uma posição se refere, e por isso seria impossível descobrir a permutação resultante da distribuição.
Assim sendo, uma transformação desses valores reais para inteiros é necessário. O exemplo na figura 2 do artigo Robles et al. (2002) é utilizado para explicar como é feito esse processo de transformação.

Original vector:	1.34	2.14	0.17	0.05	-1.23	2.18
Resulting tour:	4	5	3	2	1	6

Figure 2: Tranformação de uma permutação real para inteira

O vetor original citado na imagem 2,l é um exemplo de como seria um vetor depois do processo de distribuição ser feito. Neste caso específico o autor criou valores aleatórios entre -3 e 3 associou eles a uma posição. O segundo vetor (a rota) será um vetor de números inteiros que que representam uma cidade. A transformação é feita da seguinte forma: No vetor original, será pego os valores em ordem crescente, e na posição que estes valores forem sendo pegos será posto no vetor da rota o numero das cidades em ordem crescente.
Assim, na posição do menor valor do vetor original será colocado o identificador da primeira cidade, na posição do segundo menor valor será colocado o identificador da segunda cidade e assim por diante. No final teremos um vetor de rota organizado pelo valor da sua distribuição.
A leitura do vetor da rota é feita da seguinte forma. A cidade da I-ésima posição será a I-ésima cidade a ser visitada, assim sendo o vetor organizado acima tem uma permutação 4,5,3,2,1,6 feito pelo grau do tamanho do valor da distribuição com a cidade 2 sendo a quarta cidade a ser visitada.
Para a resolução desta transformação é utilizado o Algoritmo do Radix Sort, descrito na documentação do código fonte desenvolvimento para este trabalho em (Fernando Veras, Gleidson Mendes), assim como é possível o download do mesmo.

4.2.1 Distribuição Gaussiana - Variação

A distribuição Gaussiana é uma distribuição probabilística contínua. Ela é capaz de expor média de variáveis aleatórias como um ponto no qual a distribuição converge.

4.2.2 Cálculo do fitness

Para se calcular o fitness, deve-se somar todas as distancias das arestas selecionadas. O valor resultante deste somatório será o fitness da referida permutação.
Para o cálculo da distância entre 2 pontos é utilizado o cálculo da distância euclidiana bidimensional.

$$\sqrt{(p_x - q_x)^2 + (p_y - q_y)^2}.$$

Figure 3: Distancia euclidiana para um ponto bidimensional

4.3 Problema do Lobato

O problema do Lobato é uma simples variação do do problema do caixeiro viajante, onde o custo das viagens de uma cidade de índice par para outra cidade de índice par é reduzido pela metade. A alteração no algoritmo para a

resolução do problema se dá no cálculo da distância de uma cidade a outra, em que se ambas forem cidades pares, a distância é dividida por 2:

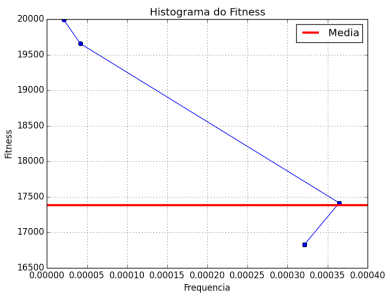
5 Resultados

Durante a execução dos testes o processo do python consumiu cerca de 25% do processador e 40 mb de memória ram. A configuração do computador utilizado nos testes seguem abaixo:

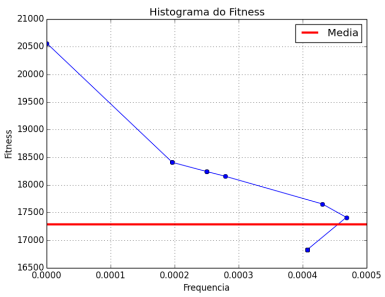
Figure 4: Computador utilizado nos testes

Realizamos testes com as seguintes instâncias: alex10.tsp, eil51.tsp e KROA100.TSP:

5.1 alex10.tsp

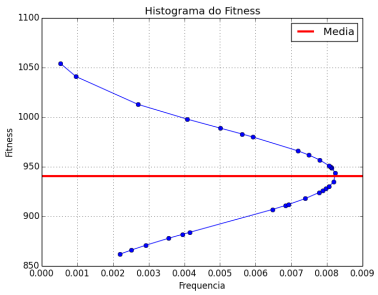


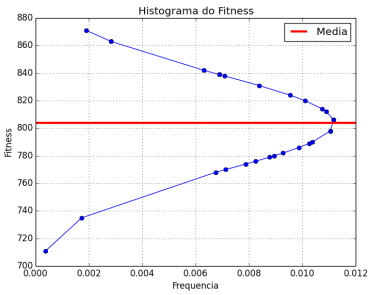
(a) Histograma do fitness para a instância alex10.tsp
Melhor: 16828
Pior: 19990
Média: 17380.2
Desvio Padrão: 1093.49108821
Média do Tempo de execução: 4.19233076572
Média das Validações: 17460.0



(b) Histograma do fitness para a instância alex10.tsp aplicando o Lobato
Melhor: 16828
Pior: 20563
Media: 17290.7333333
Desvio Padrao: 842.992524021
Média do Tempo de execução: Media: 3.96408135891
Média das Validações: 17236.6666667

5.2 eil51.tsp

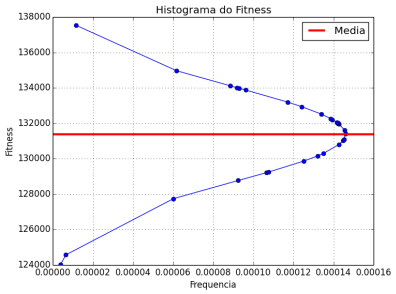


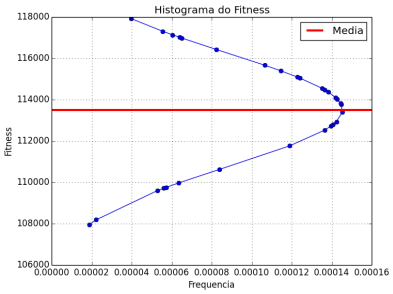


(a) Histograma do fitness para a instância eil51.tsp
Melhor: 862
Pior: 1054
Media: 940.666666667
Desvio Padrão: 48.3593033678
Média do Tempo de execução: 45.9885693153
Média das Validações: 82370.0

(b) Histograma do fitness para a instância eil51.tsp aplicando o Lobato
Melhor: 711
Pior: 871
Media: 803.866666667
Desvio Padrao: 35.643356869
Média do Tempo de execução: 47.1812816143
Média das Validações: 81260.0

5.3 KROA100.TSP





(a) Histograma do fitness para a instância kroa100.tsp
Melhor: 124020
Pior: 137533
Media: 131382.266667
Desvio Padrão: 2732.16684377
Média do Tempo de execução: 33.3240954081
Média das Validações: 35146.6666667

(b) Histograma do fitness para a instância kroa100.tsp aplicando o Lobato
Melhor: 107957
Pior: 117927
Media: 113502.366667
Desvio Padrão: 2743.75378491
Média do Tempo de execução: 35.6428841432
Média das Validações: 37033.3333333

5.4 Análise

Com estes resultados obtidos, pode-se perceber que o programa está de fato tentando achar a menor distância possível tentando ligar uma única vez cada cidade. Em todos os gráficos podemos perceber que os valores tendem a ficar bastante próximos a média das execuções.

O segundo gráfico apresentado em cada exemplo mostra os resultados obtidos utilizando a alteração para o problema de Lobato. Percebe-se que esta alteração reduz o valor do melhor fitness encontrado e da média de fitness, porém o tempo de execução é ligeiramente maior nos casos com maior número de cidades e foi menor no exemplo com 10 cidades.
Apesar destes resultados obtidos, o algoritmo não está conseguindo encontrar melhores candidatos por causa da variação dos indivíduos. No código fonte, tem-se 3 tipos de variações, mas somente a variação que está em uso obteve os melhores resultados em menos gerações. A variação apresentada, não consegue criar um modelo probabilístico totalmente correto, fazendo com que as distribuições não tendem a um valor correto, assim sendo o que poderia ser uma população ótima em uma geração, poderia se tornar um pouco pior numa próxima por uma alteração indevida nas distribuições.
Deve-se buscar um modelo de variação que melhor elabore um modelo probabilístico para a obtenção de resultados melhores

6 Conclusão

O programa desenvolvido produz soluções viáveis para o problema do caixeiro viajante, porém seu desempenho fica aquém do esperado. Utilizando para comparação a implementação de computação evolutiva proposta na documentação do framework inspyred (Aaron Garrett, a) foi obtido um tempo de execução semelhante, porém resultados bem melhores. Não foi possível fazer a comparação com os resultados da literatura utilizada para a formulação dessa solução, pois utilizava arquivos de entradas diferentes, para 500 cidades enquanto utilizamos um arquivo de entrada de no máximo 100 cidades.
O variador, que é a parte do código que gera novos indivíduos não está seguindo o modelo probabilístico corretamente, e por isso a evolução não está ocorrendo como deveria. O modelo de EDA fornecido pelo framework não estava pronto para receber instâncias do TSP e por isso foi necessária a criação de um variador, o variador criado não possui o desenvolvimento de um modelo probabilístico correto.
Entretanto foi desenvolvida uma solução que consegue procurar pelo resultado ótimo, em um tempo dentro de 1 minuto de execução, mesmo para instâncias de 100 cidades.

References

Aaron Garrett. http://pythonhosted.org/inspyred/.

Aaron Garrett. https://pypi.python.org/pypi/inspyred.

Fernando Veras, Gleidson Mendes. https://github.com/gleidsoncosta/tsp.git.

Python Software Foundation. https://www.python.org/.

Robles, V., de Miguel, P., and Larranaga, P. (2002). Solving the traveling salesman problem with edas. In *Estimation of Distribution Algorithms*, pages 211–229. Springer.